ARISTOTLE UNIVERSITY OF THESSALONIKI, GREECE
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

FINAL PROJECT THESIS

Rodrigo Lorente Sanjurjo

ANALYSIS AND DESIGN OF CONVERTERS IN MATLAB

Supervisors:
Dr. A. Hatzopoulos, professor, ECE Dept, Electronics Lab.
Dr. K. Papathanasiou, lecturer, Physics Dept., Electronics Lab.

R. Lorente Sanjurjo is a student of Telecommunication Engineering of the University Carlos-III, Madrid, Spain, studying as an “Erasmus Student” for the academic year 2009-2010, at the Department of Electrical and Computer Engineering Department of Aristotle University of Thessaloniki, Greece. The work was performed in the collaboration framework of the Electronics Lab of ECE Dept, with the Electronics Lab. of the Physics Dept, Aristotle University.

Thessaloniki 2010

El presente proyecto ha sido desarrollado en ámbito del convenio Erasmus 2009-2010, en colaboración entre los laboratorios de electrónica de los departamentos de Ingeniería Eléctrica y Computación, y de Física de la Universidad Aristotelio de Tesalónica.

Los supervisores fueron, por el Departamento de Ingeniería Eléctrica y Computación el catedrático Alkiviades Hatzopoulos, y por el Departamento de Física el profesor Konstantinos Papathanasiou. Y como cotutor en la UCIIIM Susana Patón del Departamento de Tecnología Electrónica.

El proyecto fue presentado el viernes 12 de Marzo de 2010 en la Facultad de Ingeniería Eléctrica y Computación. El tribunal estuvo compuesto por Alkiviades Hatzopoulos del Departamento de Ingeniería Eléctrica y Computación, y por Stylianos Siskos y Konstantinos Papathanasiou del Departamento de Física.

La calificación final del proyecto fue **A** (10).

AGRADECIMIENTOS

Este trabajo no habría sido posible sin la oportunidad brindada por el Programa Erasmus de intercambio, que me ha permitido conocer nuevos aspectos y puntos de vista del ámbito académico universitario, y por qué no decirlo también, visitar un país tan emblemático como Grecia, que va mucho más allá de la archiconocida Acrópolis.

De igual modo he de agradecer al profesor Konstantinos Papathanasiou por sus buenas ideas a la hora de solventar los problemas que han surgido a lo largo de estos meses, así como al profesor Theodoros Laopoulos y demás personal del Departamento de Electrónica y Computación de la Escuela de Física de la Universidad Aristotelio de Tesalónica.

También es de recibo recordar al resto de estudiantes (y no estudiantes, pero como si lo fueran) Erasmus, sin los cuales esta estancia se habría hecho mucho más difícil y aburrida: a Dora y Marisa por acogerme de refugiado, a Quique Nuxx por ser tan buen anfitrión (que aprenda la Preysler), a Juankar por esos viajes por lugares inhóspitos de Grecia, Turquía y Bulgaria…y en general a buena parte de la colonia de Erasmus en Tesalónica, no me hagáis nombraros a todos que luego hay que traducirlo.

Pero este proyecto no es más que la última etapa de un viaje mucho más largo, a donde no podría haber llegado sin la gente de la Universidad Carlos III de Madrid, en especial Uge (cuando no está rebotado) y Pepe (cuando no está pidiendo muñecos de Miku).

Obviamente también a mi familia que siempre está ahí a pesar de todo y lo difícil que puede llegar a ser tratar conmigo a veces.

Y sobre todo a Alex, por haber estado todos estos años apoyándome, al principio desde lejos, y aguantándome con mis cosas y mí asombrosa capacidad para sacarte de quicio, pero ten por seguro que todo esto no habría sido igual sin ti.

ACKNOWLEDGMENTS

This work would not have been possible without the opportunity provided by the Erasmus exchange program, which allowed me to learn new aspects and points of view of the academic field, and, why not say it too, to visit a so emblematic country like Greece, which is far beyond the well-known Acropolis.

Likewise I have to thank to Lecturer Konstantinos Papathanasiou for his good ideas when solving the problems that have arisen during these months, and to Proffesor Theodoros Laopoulos and other staff of the Department of Electrical and Computer, School of Physics, Aristotel University of Thessaloniki.

It is also necessary reminding the other Erasmus students (and no students, but as they are), without whom this stay would have made much more difficult and boring: to Dora and Marisa for receiving me as refugee, to Quique Nuxx for being such a good host (Preysler, learn from him) to Juankar for these trips by hostile places in Greece, Turkey and Bulgaria ... and in general, to many of the colony of Erasmus in Thessaloniki, do not make me nominate all of you that then I have to translate.

But this project is only the latest stage of a much longer journey, where I could not have come without the people from the Carlos III University of Madrid, especially Uge (when he is not angry) and Pepe (when he is not shopping Miku dolls).

Obviously also to my family who is always there despite everything and how hard it can be dealing sometimes with me.

And especially to Alex, for being supporting me all these years, at first from afar, and suffering me with my things and my amazing ability to get you mad, but you have rest assured that this would not have been the same without you.

INDEX

0 RESUMEN

0.1 PRESENTACIÓN Y JUSTIFICACIÓN

Con cada vez mayor frecuencia se dice que el "mundo se está volviendo digital". Y no sin cierta razón, a diario se puede ver cómo, a causa de la evolución del procesado de señales, lo que hasta hace unos años era analógico ahora cada vez más y más se convierte en digital.

Según [1] el mundo del procesado de señales se puede asemejar al modelo evolutivo en el que los mejores "organismos" sustituyen a los menos óptimos. En el tema que nos concierne, se puede apreciar como en muchos casos una solución originalmente mecánica, evoluciona a una eléctrica, y finalmente a "*estado sólido*". Una vez que la solución ha derivado en un circuito integrado, la evolución lleva a una optimización de los recursos y rendimiento, pasando de analógica, a digital y llegando a una solución software programada.

Un claro ejemplo de esto es el procesado de audio, originalmente completamente mecánico (fonógrafos), luego magnético (cassette), digital (CD) y hoy en día mediante memorias flash para su almacenamiento.

En la Figura 1.1 se consideran las diferentes técnicas de procesado de señales como capas dentro del mundo real. Además de representar el hecho de que muchos fenómenos que se quieren observar y controlar comienzan como señales analógicas que pasan a digitales para ser procesadas.

Esto podría dar a entender que los circuitos analógicos y conversores de datos se irán relegando a un segundo lugar con el paso del tiempo, pero no se puede olvidar que el mundo es analógico, y que para poder interactuar con él, se necesita poder pasar esas señales analógicas a formato digital, por lo que la importancia de los conversores, en este caso analógico-digitales, es y será de suma importancia.

Otros modelos del mundo del procesado de señales corroboran el hecho de que es imposible intentar pasar por alto el aspecto analógico de las señales y centrase en el digital, ya que a medida que uno de los dominios interiores se "expande" y lo que inicialmente era analógico, pasa a ser digital y luego software, los dominios exteriores tenderán a crecer proporcionalmente cubriendo la aparición de nuevas aplicaciones. En definitiva, mientras el

mundo del procesado de señales está evolucionando constantemente, cada vez surgen nuevas oportunidades analógicas, incluso a mayor velocidad que a la que la vieja tecnología analógica se convierte en digital.

Por todo esto, como elemento de transición entre los dominios analógico y digital, los conversores de datos se presentan como parte fundamental del procesado de señales y, por lo tanto, del reciente pasado, presente, y futuro de una buena parte del desarrollo de la humanidad.

0.2 ÁREAS DE ESTUDIO

Este proyecto, como ya se ha podido suponer, intentará proporcionar una mejor comprensión de los conversores de datos, más concretamente en la representación matemática y la codificación de no idealidades del conversor.

Como herramienta de programación se usará el entorno MATLAB, sobre el que se realizará la codificación y el análisis del comportamiento de los conversores añadiéndoles diversas no linealidades, aprovechando la simplicidad, claridad, y extensibilidad que proporciona este entorno.

En resumen, se va a estudiar el campo de los convertidores, específicamente A / D, utilizando para ello MATLAB.

0.3 OBJETIVOS GENERALES

Una vez asumida la importancia y presencia de los conversores de datos en nuestro mundo, se puede entender mejor por qué se ha decidido realizar este proyecto. Si bien resulta imposible, debido a la magnitud del campo a tratar, abarcar en profundidad todos los aspectos de los conversores de datos, este trabajo puede servir como punto de partida para otros que pretendan profundizar más en ciertas características o como apoyo en proyectos más amplios en los que se utilicen ADCs.

En concreto, este trabajo pretende mostrar, de una manera fácil y sencilla, las características y el comportamiento de un ADC según una serie de parámetros introducidos, prestando especial

atención a las consecuencias de las no idealidades inherentes a cualquier circuito real, tales como offset, no linealidad, error de ganancia, DNL, INL, etc.

Para ello se utilizará el entorno MATLAB, aprovechando su potencial en los cálculos con matrices, y que gracias a su amplia implantación en el mundo académico y comercial facilitará el desarrollo de proyectos relacionados. Aprovechando la herramienta gráfica de MATLAB, además de la salida convertida y de los datos numéricos para el estudio de las no idealidades, se puede ver gráficamente la función de transferencia del conversor y el comportamiento de la salida en función de la entrada, lo que facilitará en gran medida el estudio de los datos.

Los resultados de este proyecto final pretenden ser útiles para el análisis y simulación de circuitos de señales analógicas y mixtas en aplicaciones de transceptores modernos. Resultando útil en el co-diseño, simulación y análisis de circuitos integrados que incluyan conversores A/D, a modo de caja negra, se puede estudiar el comportamiento del circuito general, según posibles no idealidades o mal funcionamiento del conversor, y así poder contrarrestar dichos errores y que el circuito no se vea afectado.

De igual manera, el código implementado pretende resultar de ayuda en tareas educativas, para enseñar a alumnos pimerizos las consecuencias del mal funcionamiento de determinados componentes circuitales, en este caso los ADCs.

0.4 TRABAJO REALIZADO

Tras el repaso a la historia y los diferentes tipos y arquitecturas de ADCs se comienza la explicación del trabajo realizado.

Programar un conversor con un bucle resultaría extremadamente sencillo a la vez que ineficiente, únicamente habría que recorrer todos los valores de entrada y compararlos uno a uno con los umbrales de conversión de manera similar a como se haría en un algoritmo de Aproximación Sucesiva (ver apartado 2.3.2). En este caso lo que se hará será realizar todas las comparaciones simultáneamente mediante matrices, más semejante a un algoritmo de Conversión Directa o Conversor Flash (ver apartado 2.3.1) debido a la velocidad de conversión y al aumento exponencial de los cálculos por cada nuevo bit de resolución.

Lo más importante a la hora de obtener el conversor es establecer los umbrales de conversión, los valores que diferenciaran un valor de salida de otro. Estos umbrales se generan combinando distintos valores de desplazamiento de los umbrales del caso ideal y las pendientes de la función de transferencia entre dos umbrales consecutivos. Como caso ideal se entiende el conversor en el que el valor de salida aumenta linealmente con cada múltiplo de LSB .

El valor de LSB es $\mathrm{V_{REF}}/2^{\mathrm{n}}$ donde $\mathrm{V_{REF}}$ es el valor de la tensión de referencia aplicada al conversor (que multiplica a la señal de entrada y representa el máximo valor de tensión posible a la salida) y n es el número de bits con los que se realiza la conversión. A este valor se le conocerá a partir de ahora como $\mathrm{V_{LSB}}$.

A la hora de ejecutar el programa, se puede definir un parámetro "*steps*". Para usar este parámetro se ofrecen dos opciones distintas. Una sería usar un valor único que desplazaría tantos $\mathrm{V_{LSB}}$ como se indique a todos los umbrales del caso ideal. La otra opción sería usar un vector de tantos valores como umbrales tenga el conversor (es decir, 2^{n}), donde cada umbral del caso ideal se desplazará según el valor correspondiente.

El segundo parámetro que se puede variar es "*slopes*", que define la pendiente de la función de transferencia. En el caso ideal la pendiente es 1, es decir, que cada incremento en un múltiplo de $\mathrm{V_{LSB}}$ de la tensión de entrada, supone el aumento en una unidad del valor de la salida digital. Obviamente el valor de "*slopes*" tiene que ser mayor que 0, y de igual manera que para el parámetro "*steps*" se puede utilizar un vector de 2^{n} valores que represente la pendiente de cada uno de los intervalos o escalones.

Una vez calculados estos umbrales se procede a la generación de unas matrices con las que se realizarán las comparaciones que darán lugar a los distintos valores de salida, que dependerán de la cantidad de posiciones a "1" de la matriz final (siendo necesarios algunos ajustes y consideraciones en el caso de que haya situaciones de Umbrales Negativos debido al offset (ver apartado 4.3.1.1) o Burbujas (ver apartado 4.3.1.2).

Como información complementaria, el programa calcula una serie de no-idealidades del conversor generado. Más concretamente proporciona los valores de error de offset, error a fondo de escala, error de ganancia, DNL e INL (ver secciones 3.2.5 y 3.2.6). Todos los valores

están expresados en unidades de V_{LSB} y se calculan a partir de la mejor aproximación lineal de los valores de los umbrales del conversor.

Para facilitar el estudio y análisis de los resultados obtenidos se ha implementado una función que permite generar la representación gráfica del conversor y la relación entre la señal de entrada y la salida calculada [*Anexos II y III*].

Como es de suponer, el código final es lo que es después de un largo proceso en el que se tomaron diversas decisiones a la hora de implementarlo y se solucionaron una serie de dificultades, dando lugar a diversas versiones para hacer frente a un mismo problema. Como complemento se han incorporado una parte de las versiones alternativas que quizá puedan ser útiles en otras circunstancias, como el localizador de Burbujas.

1 INTRODUCTION

1.1 PRESENTATION AND JUSTIFICATION

With increasing frequency it is said that the "world is becoming digital". And not without certain reason, every day it can be seen how, because of the evolution of signal processing, what until recently was analog now more and more becomes digital.

According to [1] the world of signal processing may resemble to evolutionary model in which best "organisms" replace less optimal ones. In the subject matter hereof, it can be seen as in many cases an originally mechanical solution evolves into electrical, and finally into solid-state. Once the solution has resulted in an integrated circuit, the evolution leads to an optimization of resources and performance, passing through analog, to digital and reaching a programmable software solution.

A clear example of all this is the audio processing, originally completely mechanical (phonograph), after magnetic (cassette), digital (CD) and today with flash memories for its storage.

In Figure 1.1 are considered the different signal processing techniques as layers within the real world. In addition, to represent the fact that many phenomena that want to be observed and controlled, begin as analog signals that pass to digital for being processed.

Figure 1.1: *The "Layers" of Signal Processing*

This might imply that the analog circuits and data converters will be relegated to a second place over time, but it cannot be forgotten that the world is analog, and to interact with him, it is needed to transform those analog signals to digital format, so the importance of the converters, in this case analog-digital, is and will be extremely important.

Other models in the world of signal processing confirms the fact that it is impossible try to ignore the analog aspect of the signals and focus on the digital one, and that is because as one of the internal domains "expands", and what was initially analog becomes digital, and then software, external domains will tend to grow proportionately to cover the emergence of new applications. In short, while the world of signal processing is constantly evolving, every time new analog opportunities arise, even faster than the old analog technology that becomes digital.

For all this, as a transition element between analog and digital domains, data converters are presented as essential part of signal processing and, therefore, of the recent past, present, and future of a large part of the human development.

1.2 STUDY AREAS

This project, as it could be already expected, will try to provide better understanding of data converters, more specifically in the mathematical representation and coding of non idealities of the converter.

As programming tool it will be used the MATLAB environment, with which will carry out the coding and the analysis of the behavior of the converters by adding diverse nonlinearities, taking advantage of the simplicity, clarity, and extensibility that provides this environment.

Summarizing, it is going to study the converters field, specifically A/D, using for it MATLAB.

1.3 GENERAL OBJETIVES

Once assumed the importance and presence of data converters in our world, it may be understood better why it was decided to undertake this project. While it is impossible, given the magnitude of the field to be treated, cover in depth all aspects of data converters, this

work may serve as a starting point for others who wish to delve deeper into certain features or as support in larger projects where the ADCs are used.

Specifically, this paper aims to show, in an easy and simple way, the characteristics and behavior of an ADC according to a number of parameters introduced, paying special attention to the consequences of non-idealities inherent in any real circuit, such as offset, nonlinearity, gain error, DNL, INL, etc.

For this it will be used the MATLAB environment, taking advantage of its potential in the matrixes calculus, and that thanks to its widespread deployment in business and academia fields it will facilitate the development of related projects. Taking advantage of MATLAB graphics tools, in addition to the converted output and the numerical data for the study of non-idealities, it can be seen graphically the transfer function of the converter and the behavior of the output according to the entry, thus facilitating largely the study of the results.

The results of this final project aim to be useful for analysis and simulation of analog and mixed signal circuits of modern transceiver applications. It would be useful in co-design, simulation and analysis of integrated circuits including A/D converters, using it as a “black box”, it can be studied the behavior of the general circuit, depending on non-idealities or malfunction of the converter, so those mistakes can be counteracted and that the circuit does not be affected.

Similarly, implemented code aims to be of assistance in educational tasks to show to novice students the consequences of bad operation of certain circuitry components, in this case the ADCs.

2 THEORY OF THE ADCs

2.1 THEORETICAL BASIS

The analog-to-digital conversion (also known as ADC, A/D or A to D) consists in the transformation of analog signals into digital signals, in order to facilitate processing (encryption, compression, etc.), and make the resulting signal (digital) more immune to noise and other interferences to which the analog signals are more sensitive [2], [3], [4], [5].

This basically consists in perform periodically measures of the amplitude (voltage) of a signal, rounding the values to a finite set of preset voltage levels (known as quantization levels) and registering them as whole numbers into any type of memory or support.

Steps that are performed in an ADC:

- **Sampling:** it is to take periodic samples of the wave amplitude. The speed, at which those samples are taken, i.e. the number of samples per second, is what is known as sampling frequency.

- **Hold:** the samples taken must be retained by a holding circuit, long enough to allow the assessment of their level (quantization). From a mathematical point of view this process is not contemplated, because it is a technical resource due to practical limitations, and therefore it lacks of mathematical model. Usually a capacitor is used to store the input voltage, and it is used an electronic switch or gate to disconnect it from the entrance.

- **Quantization:** the voltage level of each of the samples is measured. It consists on assigning a value range of an analyzed signal to a single level of output. Even in its ideal version, it adds, as a result, an unwanted signal to the input signal: quantization noise.

- **Coding:** it is to translate the values, obtained during the quantization, to binary code. It has to keep in mind that the binary code is the most used, but there are other types of codes that there are also used.

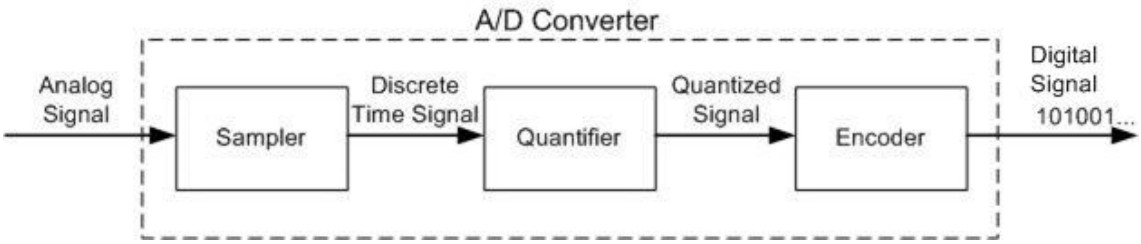


Figure 2.1: *Conversion processes in an ADC*

An analog signal is one whose amplitude can in principle take any value, that is to say, its level in any sample is not limited to a finite set of predefined levels like the quantized signals.

This does not mean that they, in practice, are signals of infinite precision (a widespread misconception): all real analog signals have a noise that translates into an uncertainty range (from a sample of an analog signal into a given instant, it is impossible to determine the exact value of the sample within a range of uncertainty introduced by the noise).

In contrast, a digital signal is that one whose dimensions (time and amplitude) are not continuous but discrete, which means that the signal must necessarily take a certain fixed values on predetermined discrete moments too.

Therefore, analog signals are not differentiated from digital signals on their accuracy (precision, that it is finite in both analog and digital) or in the fidelity of their waveforms (distortion). It is often easier to obtain precision and to preserve the waveform of the original analog signal (within the limits of accuracy imposed by the noise that exists before his conversion) in the digital signals, than in those which come from analog supports that are characterized typically with a low signal to noise ratio in comparison.

2.1.1 Advantages and disadvantages of digitizing a signal

Advantages

- When a digital signal is attenuated or experience mild disturbances, it can be reconstructed and amplified by signal regeneration systems.

- There are systems to detect and correct errors, which are used when the signal reaches the receiver, then checks (using redundancy) the signal, first to detect an error, and, in some systems, correct any or all the errors found previously.

- Easiness for processing the signal. Any operation is easily realizable through any editing or signal processing software.

- The digital signal allows multigeneration (number of processes of compression/decompression that can be applied to an information without diminishing its quality) infinitely without quality loss.

- It is possible to apply data compression techniques lossless or data compression techniques with losses, based on perceptual coding, much more efficient than with analog signals.

Disadvantages

- It is needed a previous analog-to-digital conversion and the subsequent decoding at the time of receipt.

- If there is not employed a sufficient number of quantization levels in the process of digitization, the resulting signal to noise ratio is reduced compared to the original analog signal that was quantified. This is a consequence of that the signal known as quantization error, that the process of quantification introduces, is always more potent than the noise of the original analog signal, in which case it also requires the addition of a noise called *dither* (see Section 2.2.10) even stronger to ensure that the error is always a white noise and not a distortion. In cases where enough levels have been used for quantization, the signal to noise ratio of the original signal is kept essentially because the quantization error will be below the noise level of the signal that was quantized.

- It is necessary to use always a low-pass analog active filter over the signal to be sampled in order to avoid the phenomenon known as *aliasing* (see Section 2.2.9). Also during the reconstruction of the signal in the posterior D/A conversion, it is also

necessary to apply an analog active filter of the same type (low-pass) known as filter of reconstruction.

2.2 CONCEPTS AND CHARACTERISTICS

In this section are discussed some of the most common parameters and concepts when working with converters [2], [5], [6], [7], [8], [9], [10].

2.2.1 Resolution

The resolution of a converter indicates the number of discrete values in what the entire range of analog values can be transformed. These values are typically stored in binary, so the resolution is usually expressed in bits. Therefore the number of "levels" available for conversion is usually power of two. For example, an ADC with 3-bit of resolution can encode an analog input in one of 8 levels ($2^3 = 8$).

It can be expressed in volts per step (Q), knowing the range of the reference voltage (in this work between 0 and V_{REF}) and the number of bits used to encode:

$$Q = \frac{V_{REF}}{2^{nbits}-1} \tag{2.1}$$

In practice, the useful resolution of a converter is limited by the better signal to noise ratio (SNR) that can be achieved for a digital signal. An ADC can resolve a signal only till a limited number of bits of resolution, known as the effective number of bits (ENOB).

2.2.2 Sampling Rate

The analog signal is continuous in time, it must be defined the speed at which the digital values are sampled from the analog signal, what is known as the sampling rate or sampling frequency.

To retrieve all information of the converted signal, using interpolation, the sampling frequency at which the conversion was carried out must satisfy the sampling Nyquist-Shannon theorem (it must be at least twice the highest frequency present in the analog signal, or what is the same, the double of the bandwidth).

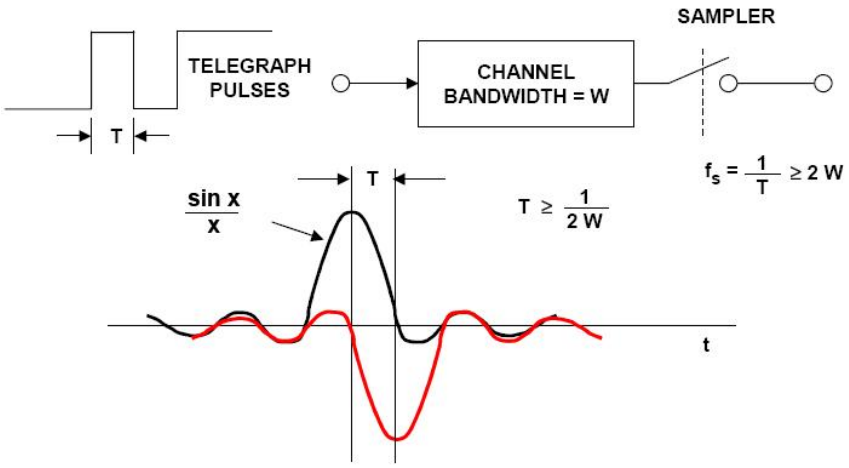


Figure 2.2: *Nyquist's Classis Theorem (1924)[2]*

2.2.3 Quality parameters

All these parameters are expressed in dB

- **Signal to Noise Ratio (SNR):** it is the margin between the power of the signal that is transmitted and the power of the noise that corrupts it.

$$SNR = \frac{P_{signal}+P_{noise}}{P_{noise}} \tag{2.2}$$

- **Signal-to-Noise and Distortion (SINAD):** it is the parameter that measures the quality of the signal against disturbances such as noise, interference or distortion.

$$SINAD = \frac{P_{signal}+P_{noise}+P_{distorsion}+P_{interference}}{P_{noise}+P_{distorsion}+P_{interference}} \tag{2.3}$$

- **Effective Number of Bits (ENOB):** if the converter can represent signals below the noise level, lower bits represent only noise and contain no useful information. ENOB specifies the number of bits in the digitized signal above the noise level. Usually:

$$ENOB = \frac{SINAD-1.76}{6.02} \tag{2.4}$$

- **Spurious-Free Dynamic Range (SFDR):** it is the relationship between the mean square values of the main frequency (the one which has a higher signal component) and the value of the second largest harmonic component.

2.2.4 Quantization Error

The quantization error is due to the finite resolution of an ADC, and it is inevitable in all types of converters. The magnitude of this error in the sampling instant is between 0 and $\frac{1}{2}LSB$.

2.2.5 Offset Error, Full-Scale Error and Gain Error

The Offset Error is the output that shows the converter when the input is the voltage corresponding to the digital output "0".

The Full-Scale Error is the difference between the input at the maximum possible digital output and the ideal case.

Gain error is the difference between the response of the converter and the ideal response at Full-Scale once made Offset Error zero.

2.2.6 Non-linearity

All ADCs suffer from non-linearity errors caused by their physical imperfections, this causes that their outputs deviate from a linear function (or some other function in the case of deliberate non-linear converters [3]). Sometimes these errors can be limited by calibration, or be prevented.

The most important parameters of this paragraph, and those ones in what is focused part of the results of this project are Integral Non-Linearity (INL, deviation between the midpoint of a code from the best linear approximation in $LSBs$) and Differential Non-Linearity (DNL, deviation in the width of a particular code value from $1LSB$), these two reduce the dynamic range of signals that can be digitized, thereby reducing the effective resolution of the converter.

2.2.7 Missing Codes

Missing Codes occur when there is no input voltage value that generates a determined output code, so it will never appear at the output and it disappears from the transfer function.

2.2.8 Aperture Error

The Aperture Error is a type of error generated in the components themselves and not correctable. It is the uncertainty in the input signal in the time when the signal is sampled and held, it is given from the moment that it is sampled until it is held, before moving to the conversion process. The opening error may be generated by noise or also by variations in the clock signal. This error also influences to the final characteristics of the system.

This error can be reduced if the sampling and holding time decreases. If this value is as small as possible, then there will be a big certainty that the conversion is done correctly.

2.2.9 Aliasing

As all ADCs work by sampling at discrete intervals, it is impossible to know the behavior of the input from one sample to the next one at the output. If the input signal changes slowly compared to the sampling frequency can be assumed that the value between two sampling instants will be between the two sampled values, but this cannot be assumed when the signal changes faster than the sampling rate, which is the cause of *aliasing* [11].

This phenomenon causes that frequency components outside the band of interest will be recorded as false frequency components within the band of interest. For example, a $2kHz$ sinusoidal signal sampled at $1.5kHz$ will be rebuilt as a $500Hz$ sinusoidal. So it becomes necessary to filter low-pass the signal at the input of the converter.

2.2.10 Dither

It is known as *dither* to the random noise (white noise) of low level to be added to the input signal, it is added before sampling to avoid or minimize quantization distortion, so it has to be more powerful than the quantization noise to make it white.

2.3 ARCHITECTURES AND TYPES OF CONVERTERS

Here there are some of the most common architectures and types of ADCs, although there are many others [2], [3], [8], [12].

2.3.1 Direct Conversion ADC or Flash ADC

It has a bank of comparators that sample the input signal in parallel, each for a certain range of voltage. Then, with a logic circuit, it is generated the corresponding code. It is the fastest converter, but its resolution is usually limited to 8 bits at most, and its complexity and size increase with each new resolution bit.

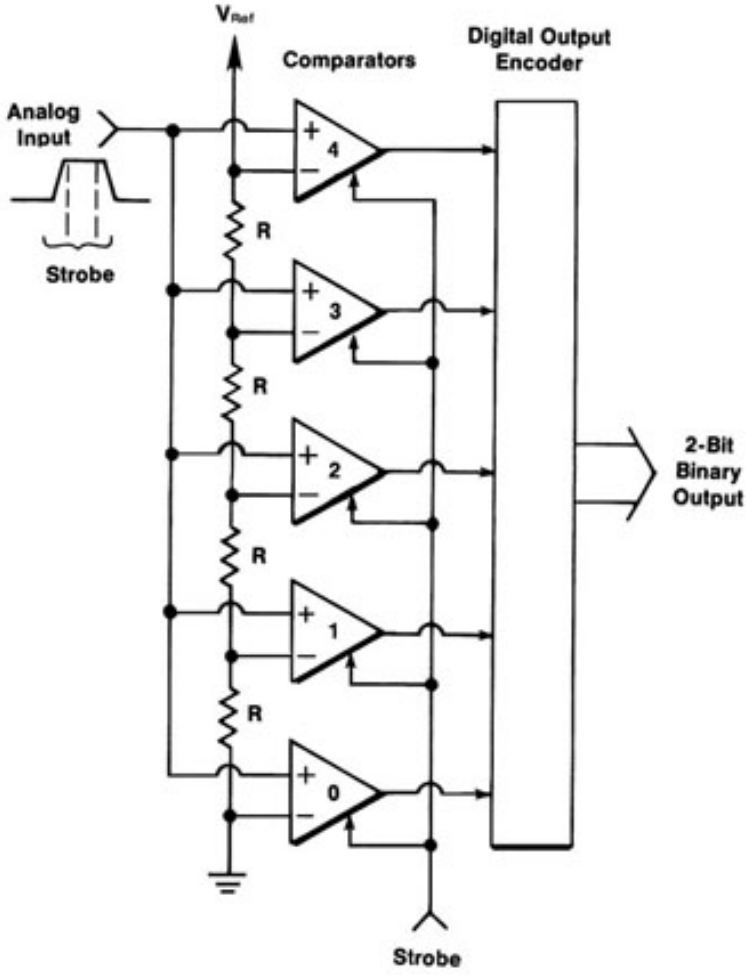


Figure 2.3: *2-Bit Flash ADC [8]*

2.3.2 Successive-Approximation ADC

The comparator will go discarding voltage ranges till find the right one. It constantly compares the value with the output of an internal digital-analog converter until the best approximation is achieved. At each step it is stored a binary value of the approximation in a successive approximation register (SAR).

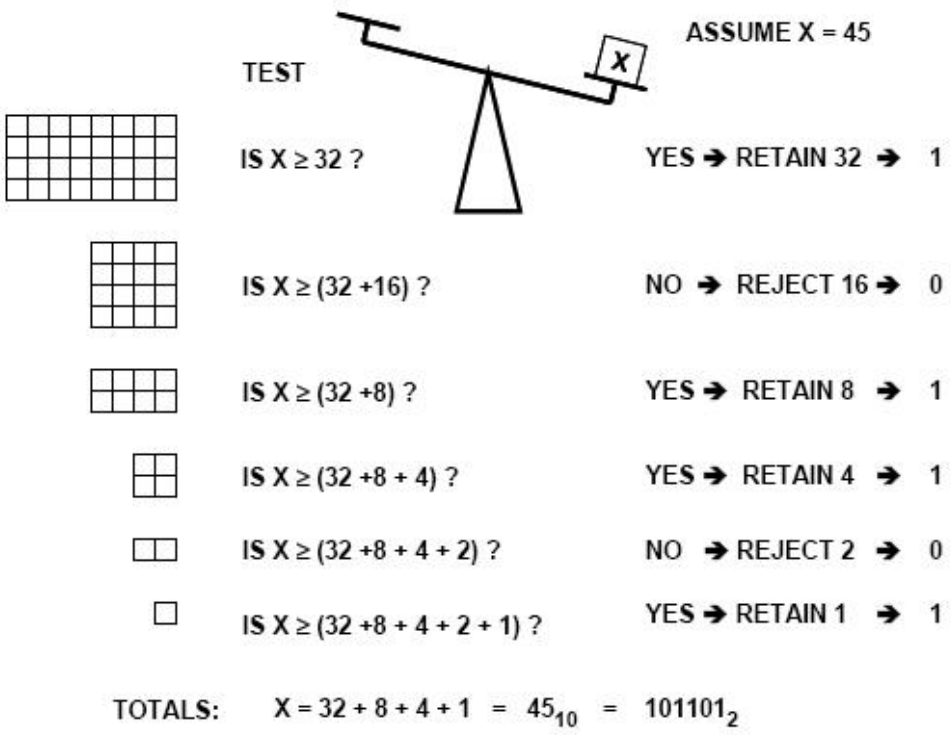


Figure 2.4: *Successive Approximation ADC Algorithm Analogy Using Binary Weights [2]*

2.3.3 Wilkinson ADC

It is based on compare the input voltage with the voltage produced by a load capacitor. The capacitor charges while its voltage is less than the input pulse, and then it discharges linearly, achieving a voltage ramp. At the moment that the discharge begins a pulse starts, its duration will be the duration of the discharge, so it is proportional to the input amplitude. This pulse operates a logic gate that allows the pass of a discrete number of pulses that will also be proportional to the input, in a similar way as it happened in Reeves's patent (Section 3.1.1).

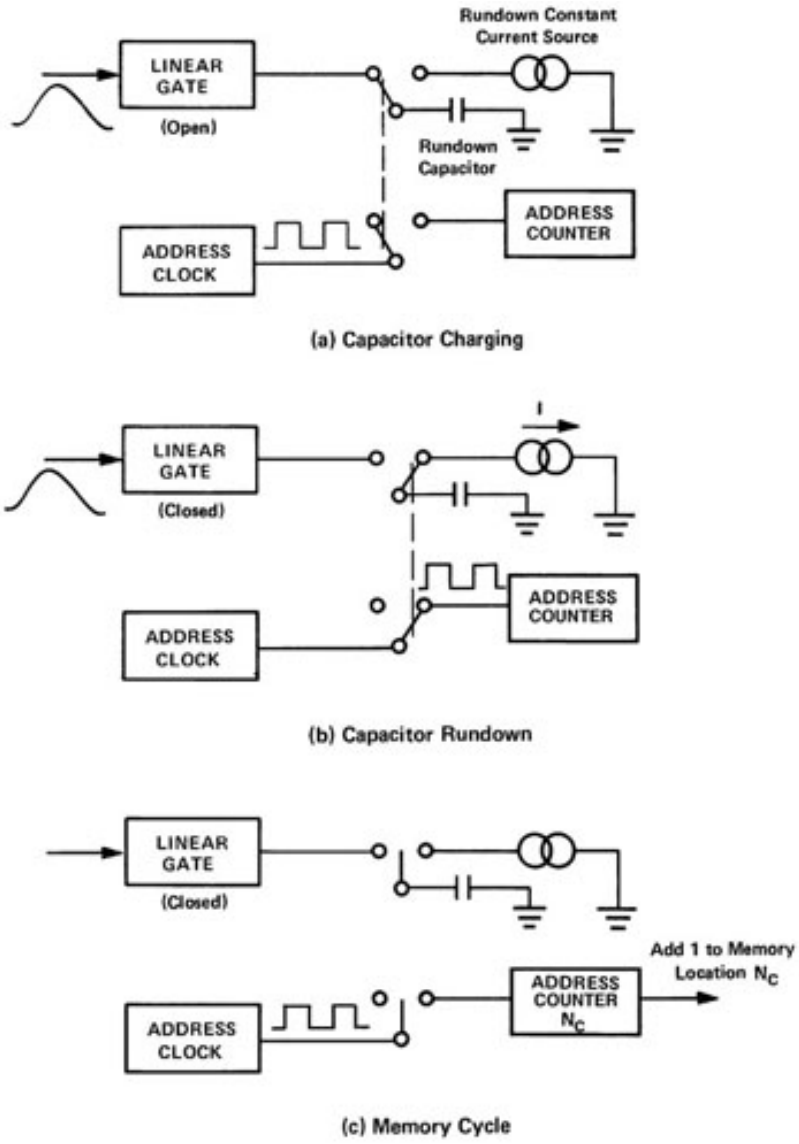


Figure 2.5: *Operation of the Wilkinson ADC [8]*

2.3.4 Sigma-Delta ADC (Σ-Δ)

Usually the signals are sampled at the minimum rate needed by economy, but this makes that the introduced quantization noise is white noise across the whole operating range of the converter, to limit it to the bandwidth of the signal (thereby improving performance as resolution and SNR), it exists the oversampling, which consists in sampling at a rate much higher than the Nyquist frequency [3], [11]. This is the principle in which are based the Σ-Δ converters, which after filtering the result, it also uses a flash ADC to convert a few of the bits. The resulting signal is fed back negatively to the input of the filter, causing *noise shaping* the error of the Flash converter, removing it from the desired frequencies.

2.3.5 Pipeline ADC

Initially, a coarse conversion is done, in a second step it is obtained the difference of the input signal with a DAC which then will be adjusted to the correct value in different stages. Because of these different steps the converter has a certain delay that does not necessarily affect to its performance but there are situations that will require pay attention to this fact. It comes to be an improved successive-approximation algorithm.

2.3.6 Integrating ADC (also dual-slope or multi-slope ADC)

It applies the unknown input voltage to an integrator and it allows the voltage to ramp for a fixed time period. Then a known reference voltage with opposite polarity is applied to the integrator until the output returns to 0. The input will be based on the reference voltage, and the rise and fall periods. It is commonly used in voltmeters.

2.3.7 Ramp-compare ADC

It generates a saw-tooth signal, when it begins, the timer starts, and when the value matches with the input the timer value is reset and saved. It is the one that needs fewer transistors.

2.3.8 Time-interleaved ADC

It uses M ADCs in parallel, each one samples in a given cycle of the clock. With this it is achieved to increase the sampling rate M-times, but overall performance is reduced because of reducing the SFDR, but there are technologies to correct this.

3 HISTORY OF THE DATA CONVERTERS

3.1 ORIGINS AND EVOLUTION

3.1.1 Early History

It is difficult to determine accurately the first time that a data converter appeared and which form had. Older data lead to Turkey, in the eighteenth century, where it can be found information about a binary DAC. The DAC was not an electronic converter, but hydraulic (Figure 3.1), and consisted of a sophisticated system to measure water based on a system of nozzles of different sizes (multiples and sub-multiples of the basic unit of 1 lüle, corresponding to 36 l/min) using a reservoir that maintained a constant level of water thanks to a spillway. It is functionally an 8-bit DAC, and may be the oldest DAC in the world.

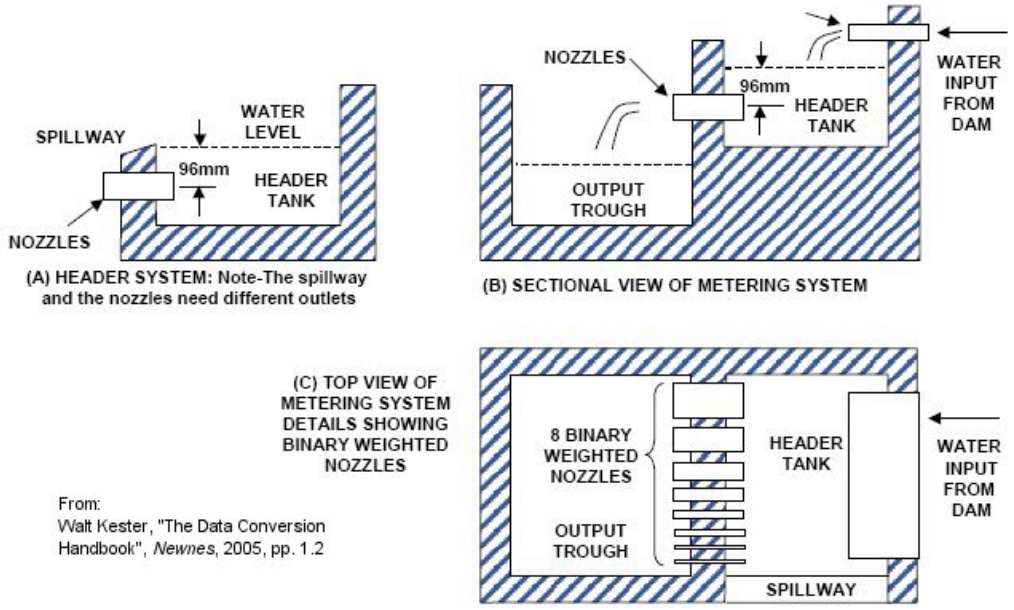


Figure 3.1: *Early 18th Century Binary Weighted Water Metering System [2]*

The main force behind the development of electronic data converters over the years has been the communications field. The telegraph led to the invention of the telephone, and the proliferation of both resulted in an increased demand for more capacity, that resulted in the need for multiplexing more than one channel in a pair of copper conductors. In the field of multiplexing are several options, the time division multiplexing (TDM) achieved some popularity, but it was frequency division multiplexing (FDM) using carrier-based systems, the

one which achieved greater success. However it was the pulse code modulation (PCM) the one that gave the importance that the data converters have today.

The Invention of PCM

Pulse code modulation first appears relatively hidden in a patent of Paul M. Rainey of Western Electric in 1921. The patent describes a method to transmit fax information encoded over a telegraph line using 5-bit PCM (Figure 3.2).

Using a photocell to capture the transparency of the material according to the intensity of light received, it generated a current that moved to a galvanometer with in turn moves a beam of light that activated one of 32 photocells, which were connected to relays. The five relay outputs were connected in such a way that generated the code corresponding to the position of the photocell. This serial information was transmitted, received, and converted to parallel format to run the lamp whose intensity was proportional to the received code.

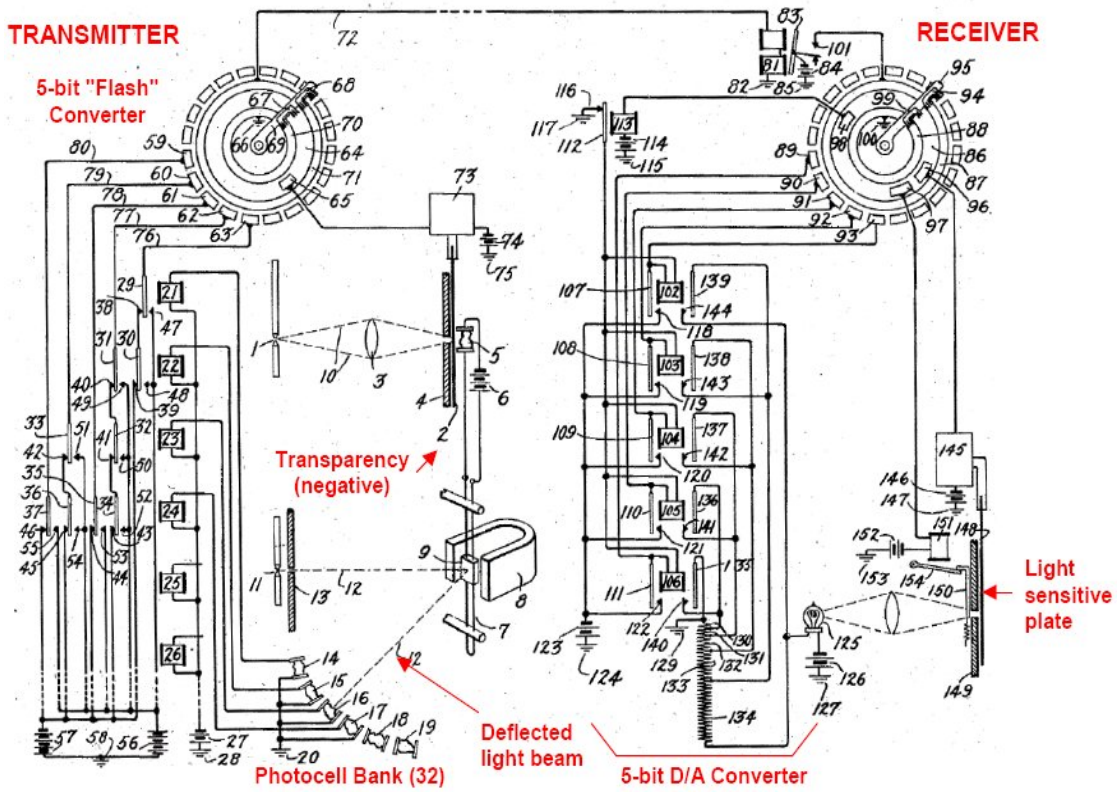


Figure 3.2: *The First Disclosure of PCM: Paul M. Rainey, "Facsimile Telegraph System", U.S. Patent 1,608,527, Filed July 20, 1921, Issued November 30, 1926 [2]*

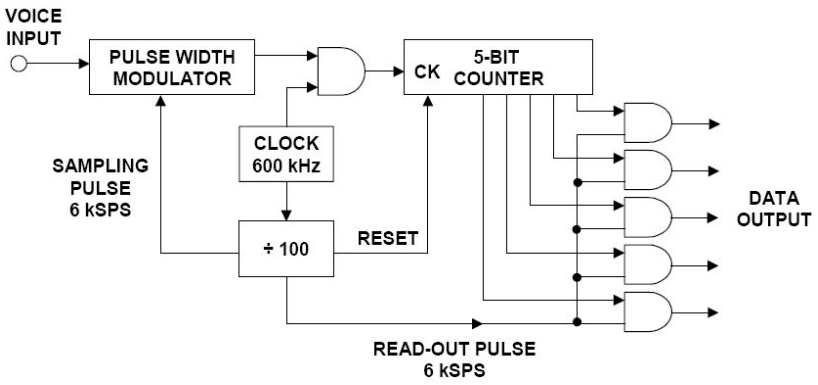
In the patent appeared several important concepts: quantization using a flash A/D converter, serial data transmission, and reconstructing the data using a D/A converter. Being these the foundations of PCM.

The PCM Patents of Alec Harley Reeves

By 1937 the frequency division multiplexing (FDM) based on vacuum tube technology was widely used in the telephone industry for long distance communications. However there were two limiting factors, noise and distortion.

Alec Harley Reeves had studied the techniques of A/D conversion using pulse time modulation (PTM). In PTM, the amplitude of the pulses is constant, while the analog information is in the duration of the pulses, which gives a greater immunity to noise. The need for a system with noise immunity similar to telegraph was what drove Reeves in 1937 to reinvent PCM with some patents which, among other things, covered the counter-based design for 5-bit ADCs and DACs, using the vacuum tube technology in its design, unlike PCM patented by Rainey.

These converters developed by Reeves are considered ones of the first all-electronic on record. Focusing on the ADC (Figure 3.3) it basically uses a sampling pulse to take the analog signal, activate a R/S flip-flop, and simultaneously start a ramp voltage. This ramp is compared with the input and when they are equal, a pulse is generated which resets the flip-flop, whose output is a pulse with a width proportional to the analog signal at the sampling instant. This pulse train with pulse width modulation (PWM), after passing an oscillator, can be easily converted into a binary word by a counter.



Adapted from: Alec Harley Reeves, "Electric Signaling System," *U.S. Patent 2,272,070*, Filed November 22, 1939, Issued February 3, 1942

Figure 3.3: *A.H. Reeves' 5-bit Counting ADC [2]*

World War II through 1948

During World War II efforts in the field of the converters were directed toward a secret system of voice known as "Project-X" that used PCM. This research led to the Successive Approximation ADC (see Section 2.3.2), the decoder of Shannon-Rack and the practical demonstration that the PCM was feasible.

One of the most significant evolutions of this period was the electron beam coding tube (Figure 3.4). The analog signal first passes through a stage of sample-and-hold, and while the "hold" interval, the beam makes a horizontal sweep. The Y-deflection for a single sweep corresponds to the value of the analog signal sampling. With a coded shadow mask it is generated the corresponding binary code. The code is registered by the collector, and the bits are generated in serial format (Figure 3.4A). Later models used a fan-shaped beam, leading to the first electronic "flash" converter (see Section 2.3.1) delivering an output in parallel (Figure 3.4B).

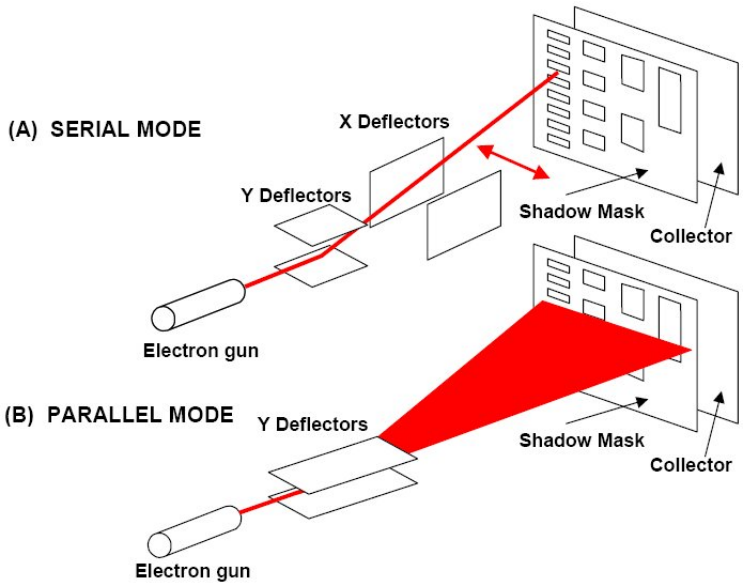


Figure 3.4: *The Electron Beam Coder [2]*

The problem with this method is that it can produce large errors if two adjacent codes are simultaneously illuminated. This is solved partly by using shadow masks with Gray code because adjacent codes differ only in one bit, so if there is a mistake, this will be of only one LSB.

3.1.2 Data Converters of the 1950s and 1960s

Commercial Data Converters: 1950s

Until the mid-1950s, converters were developed and used for specialized application. Due to the vacuum tube technology, the converters were expensive, bulky and with high energy consumption.

The digital computer pushed the commercial ADC development. The ENIAC computer development project brought the first commercially available digital computer, the UNIVAC.

In the beginning the military applications were the precursors of digital computers, but over time the possibilities in the areas of data analysis and industrial process control created a new interest in digital processing, and therefore the need for data converters. In 1953 Bernard M. Gordon, a pioneer in the field of data conversion and who worked on the UNIVAC computer, founded EPSCO Engineering. In 1954 EPSCO presented an ADC (the EPSCO "Datrac"), based on vacuum tubes, and of 11-bit and 50-kSPS, being this converter the first commercial offering of such a device, and the first commercial ADC capable of digitizing ac waveforms, such as speech.

Figure 3.5: *1954 "DATRAC" 11-bit, 50-kSPS Vacuum Tube ADC Designed by Bernard M. Gordon at Epsco [2]*

Commercial Data Converter: 1960s

If there had to talk about the greatest leap in the history of electronics, that was certainly the invention of the transistor, considered by many "the greatest invention of the twentieth century". Although it was invented in December 1947 by W. Shockley, J. Bardeen and W. Brattain [13], it was not until the mid-1950s and early 1960s that the design of circuits began to migrate from vacuum tubes to transistors, allowing a considerable reduction in size and consumption of the devices and stop aside the problems inherent in vacuum tubes (volume, fragility, consume, cost ...).

The other reason that led to the reduction of size and the consumption of data converters was the emergence of new building blocks of integrated circuits (that had evolved since its development by J. Kilby in the summer 1958) as the 7400-series of TTL and high-speed ECL, and the 4000-series CMOS logic, in addition to Zener and Schottky diodes, FETs for switches,...

During this time the most popular applications of the converters were the increasingly interest in data analysis, industrial process control, measurement, PCM, and military systems (hardware and software mainly for anti-missile ballistic systems). The ADC acquired great importance in the control of interceptor missiles of short and long range, being used high-speed ADCs in the phased array radar receiver.

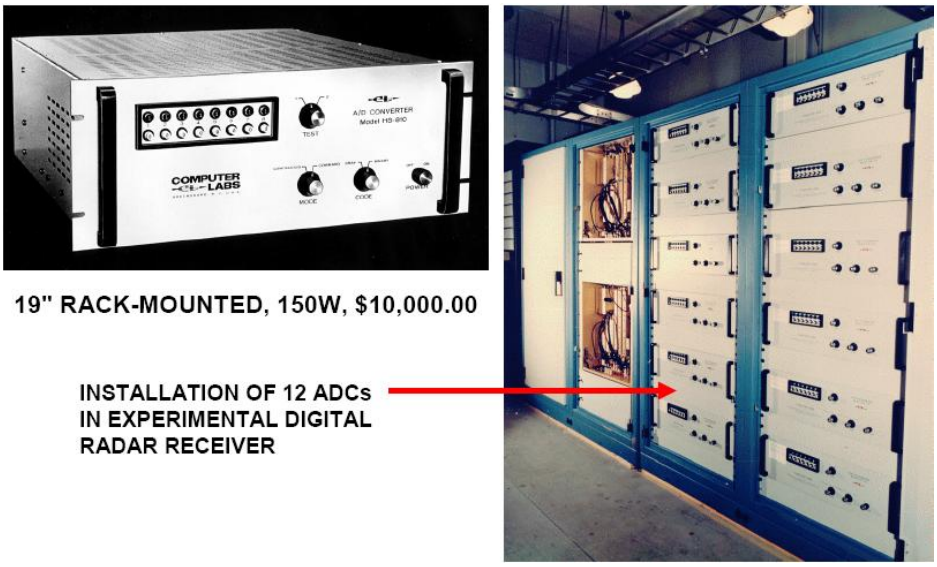


Figure 3.6: *HS-810, 8-bit, 10-MSPS ADC Released by Computer Labs, Inc. in 1966 [2]*

During this decade there were discovered and published, one way or another, all the fundamental architectures of the ADCs (for more information on types and architectures of ADCs see Section 2.3).

3.1.3 Data Converters of the 1970s

Introduction

1970 marked the beginning of one of the most important decades in what is referred to data converters. The high resolution digital voltmeters, industrial process control, digital video, the military field, medical imaging,..., represented a wide range of applications for converters. Many of these systems had used previously conventional analog signal processing techniques, and the development of low cost computing technology led to a migration to exploit the advantages of digital processing and signal analysis.

All this caused that many companies entered in the field of data converters, exploiting to the maximum all available technologies: monolithic, modular, and hybrid, with the latter two offering better features in terms of resolution and speed by allowing to use components that cannot be included in a monolithic integrated circuit (in which all components depart from the same chip of semiconductor and are manufactured jointly) as trimmable passive elements.

Monolithic ADCs of the 1970s

Although most of the converters in the early 1970s were modular or hybrid, the developers spared no efforts to produce an all-monolithic ADC [14].

After several attempts, in 1978, it was introduced the 10-bit, 25- μs AD571 SAR ADC, the first all-monolithic ADC. Later, that same year, it would see the light the SAR (Successive Approximation Register, see Section 2.3.2) that probably has been the most important ever made, the 12-bit, 35- μs AD574, with timing circuits, three-state output buffers,...

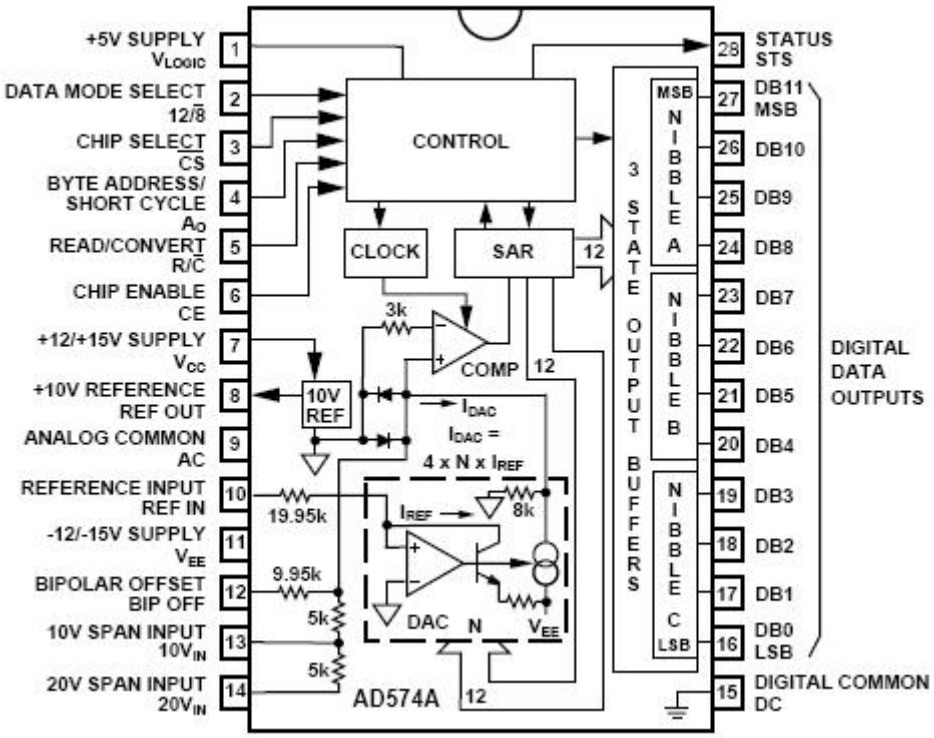


Figure 3.7: *The Industry-Standard AD574 12-Bit, 35-µs IC ADC, 1978*

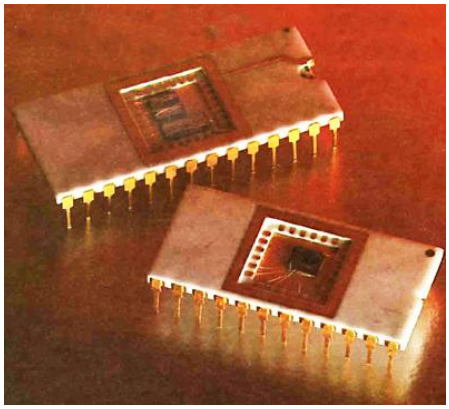


Figure 3.8: *AD574 two-chip, 1978*

At the end of this period they begin to emerge the first high-speed video flash ADCs.

Hybrid ADCs of the 1970s

The demand for cheaper data converters, compact and higher performance, meant that manufacturers focused on modular and hybrid techniques. Although the production of hybrid versions was more expensive, it allowed to performance levels unattainable for monolithic technology of the time.

A great example of this technology was the AD572 12-bit, 25- μs SAR ADC introduced in 1977, which included internal clock, voltage reference, comparator, and input buffer amplifier, being the first 12-bit ADC that passed the test standard MIL-STD-883B for military and aerospace use.

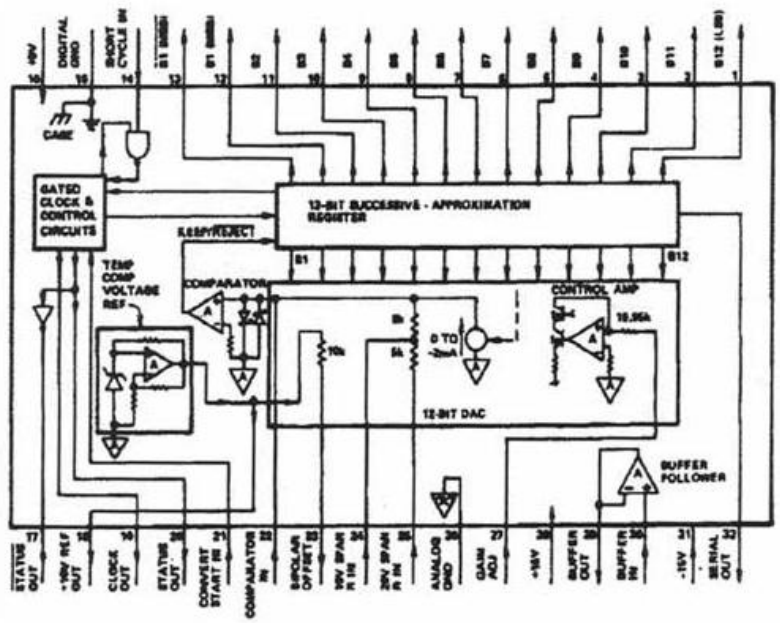


Figure 3.9: *AD572 12-Bit, 25-µs Hybrid ADC, 1977*

It should be kept in mind that at that time none of the monolithic or hybrid ADCs have an internal SHA (sample-and-hold) system, so to process ac signals was necessary to connect a SHA to the ADC.

Modular ADCs of the 1970s

The modular design allowed greater flexibility than hybrid. The modular technology emerged in the late 1960s before hybrids were made popular, and still exists today for certain products.

From this technology can highlight the ADC -12QZ 12-bit, 40-μs SAR ADC of 1972.

3.1.4 Data Converters of the 1980s

Introduction

The 1980s meant a high growth in hybrid and modular converters, driven by the needs in the fields of instrumentation, data acquisition, medical imaging, audio/video, computer graphics ...

however, as in the previous decades, the dominant drivers were the military and aerospace applications.

With the presence of low-cost microprocessors, high-speed memories, DSPs and the emergence of the IBM compatible PC, it became necessary that the ADCs were able to operate in wide ranges of frequencies, which resulted in that the converters began to take into account concepts such as signal-to-noise ratio (SNR), signal-to-noise and distortion (SINAD), effective number of bits (ENOB), noise power ratio (NPR)...

Numerous Flash ADCs arise with high-speed bipolar and CMOS technology. The 16- and 18-bit converters appeared to cope to the needs in the field of voiceband and audio signal processing.

In short, the converters provided increased resolution and functionality, including the emergence of multichannel converters.

Monolithic ADCs of the 1980s

Initially, the AD574 of 1978 was composed of two chips, but in 1985 appeared the single-chip version, which caused that the converter was definitely established as an industry standard that remains in production today.

The CMOS processes allowed include, with increasing ease, references, buffer amplifiers and sample-and-hold systems.

Although the basic architecture of the Σ-Δ ADC (see Section 2.3.4) was known since the 1950s and 1960s, the first commercial offering of a monolithic converter of this kind was in 1988, having 16-bit resolution and 20 kSPS. The increasing demand for higher resolutions (more than 20-bit) and yields, led to a rapid growth of the converters with Σ-Δ architecture.

Monolithic Flash ADCs of the 1980s

The rapidly growing digital video market generated the need for high speed converters, which led to the development of flash ADCs lower powered, resolutions between 4- and 10-bit, and high sampling rate (up to 500 MSPS).

For this it was used both bipolar and CMOS technology, the latter being the one with lower consumption at the expense of inferior performance (although with time this problem has been solved in part).

Hybrid and Modular ADCs of the 1980s

Continuing the trend of the previous decade, the demand for hybrid and modular converters peaked in the 1980s because of its better performance compared to monolithic.

3.1.5 Data Converters of the 1990s

Introduction

In this decade there occurred the final boom of the converters. Until the mid-1990s, following the trend of the previous decade, the impetus for the development of converters came from consumer applications, both audio and video, which by then began to "go digital". The communications field generated at the end of the decade a huge demand for converters of low-power, low-cost, and high-performance, for the development of cell phones, modems, and base stations.

One of the main objectives was to allow portable applications, making the subject of the consumption extremely important. Moreover, the market demanded increasingly digital and analog functionality, including the fact that there was growing applications requiring both an ADC and a DAC, which led to integrate both into a single chip, called a coder-decoder, or CODEC.

The increase in signal processing applications in the frequency-domain further emphasized the need for data converters that operate in dynamic range and ac performance. The architecture of the converters began to migrate from the flash ADCs in the decade before, to the pipeline, while CMOS technology became the most used in data converters.

Monolithic ADCs of the 1990s

During this decade, monolithic ADCs achieved the performance of hybrid and modular converters, mainly thanks to the huge reduction of parasitics in the new IC processes.

The different converters meant an improvement in certain aspects. Regarding to speed and performance highlights the AD872 12-bit, 10-MSPS BiCMOS sampling ADC in 1992.

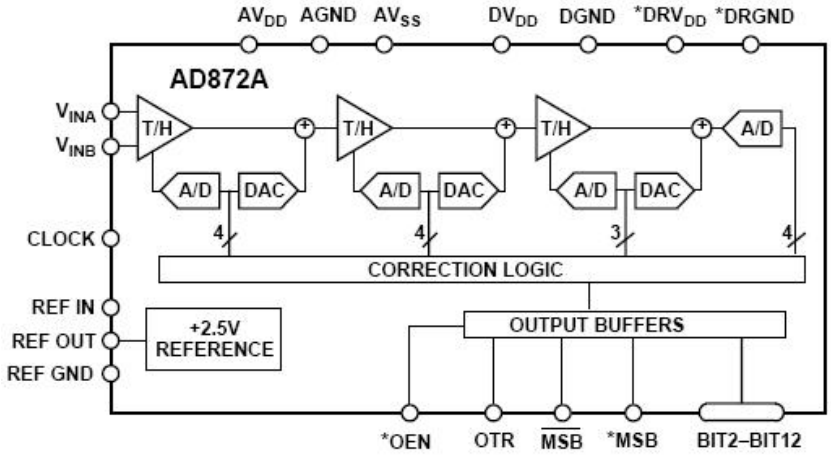


Figure 3.10: *AD872 12-Bit, 10-MSPS BiCMOS Sampling ADC, 1992*

Regarding the wide dynamic range IF-sampling ADCs, in 1995 it arose the first converter capable of achieving more than 80-dB SFDR (Spurious-Free Dynamic Range) for signals over 20-MHz Nyquist bandwidth, the 12-bit, 41-MSPS AD9042.

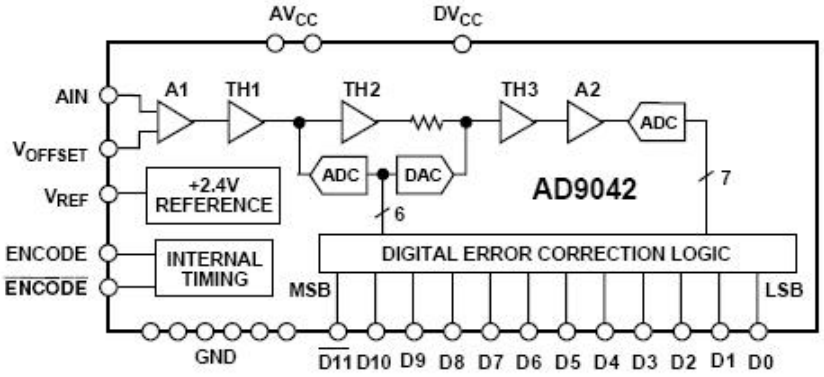


Figure 3.11: *AD9042 12-Bit, 41-MSPS XFCB Sampling ADC, 1995*

For measurement, voiceband, and audio ADCs, the selected architecture started to be the CMOS Σ-Δ ADCs, where the AD7001 was the first GSM baseband converter.

Hybrid and Modular ADCs of the 1990s

Due to improvements in the characteristics of the monolithic converters, modular and hybrid models began to lose market presence. Despite this, to the early decade, some models

emerged, but later in the 1990s, the multichip module (MCM) technology became an excellent alternative to the expensive modules and chip-and-wire hybrids.

3.2 PRESENT AND FUTURE DEVELOPMENTS

In the past 30 years there have been varying the applications/industries that led the technology development of the converters. In the early 1980s, military and aerospace applications were those that fostered the development, improving both resolution and speed. In the mid-1990s the drivers were the consumer applications as video and audio became digital. While in the late 1990s and early 2000s it took the baton the technology geared towards computers, which became a true mass market.

Today Internet and broadband communications head most of the advances in data converters, and it is at this time, further than improvements in dynamic range and bandwidth, when the market has begun to demand constant reduction in area/cost and better power efficiency, all this linked to the new portable applications.

As CMOS technology moves toward the deep submicron regime [15], process technologies offer significant increases in bandwidth at the expense of headroom.

Although the architectures of data converters have evolved a great deal over time and new architectures have emerged, it is difficult that any architecture disappears and be stop using, but it is normal that they coexist with each other, being used each one as needed. For example, Flash converters are used for applications that requires high speeds but which bear low-moderate resolution, while for applications that require a high dynamic range, such as audio, there are chosen Sigma-Delta converters, pipelined converters are used for large resolutions, like 10-14 bits, with sampling rates greater than 1MSPS...

In recent years also it has opted to interleave multiple converters to achieve higher sampling rates. However, what will lead the next generation of technology for converters, it will be the use and needs that will be required for the converters, taking advantage of the deep submicron CMOS technology:

- The digital functionality tends to be faster and cheaper than analog technology. It is cheaper to implement filtering and other signal manipulation in the digital domain,

which lead to the converter closer to the "original signal". Moreover, convert functions from analog to digital usually requires higher sampling rates for filtering. These higher speeds provided by new technologies can also be adapted to address some of the problems of low-speed converters like Successive Approximation converters.

- It is becoming more important to integrate converters with other functions. These partitioning trade-offs are becoming an important part of the design of converters.

- Once the improved performance of the converters is not reflected in an appreciable benefit for full functioning of the system, innovation is focused on reducing costs and improving power efficiency. Also, when it is reached this state of "performance saturation" often appear new applications that can benefit from higher levels of performance.

- Deep submicron CMOS allows a very high level of transistor density, so those technologies with a large number of comparators (such as Flash, which for an N-bit converter requires 2^N comparators) become again viable options in the development of converters.

- Using digital post-processing a large number of analog non-idealities can be compensated.

Making a compilation of the past 30 years:

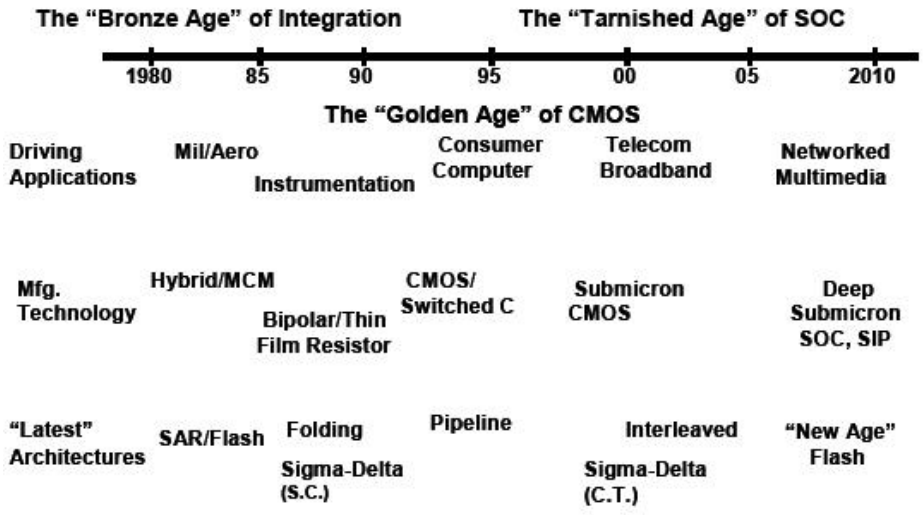


Figure 3.12: *30 Years of ADC Development [1]*

For more information about converters, both A/D and D/A, it is recommended to consult [1] and [2].

4 WORK PERFORMED

4.1 PRESENTATION

Following the review of the history and the different types and architectures of ADCs, it can begin the explanation of the performed work.

As discussed above, one of the advantages of MATLAB is its power in the calculation of matrices, so, to achieve an optimal converter, it has been tried to capitalize on this aspect. Because of that, it has been implemented a code without loops (one of the weaknesses of MATLAB) in the part of the converter.

In parallel, it has been developed a function that allows graphically displaying the transfer function of the converter in addition to the input and output signals, thereby facilitating the analysis of results and its understanding.

4.2 OPERATION OF THE CODE

To program a converter with a loop would be extremely easy at the same time that inefficient, it suffices go over all the input values and compare them one by one with the conversion thresholds in a similar way as it would be done in a Successive Approximation Converter (see Section 2.3.2). In this case what it will be made is to perform all comparisons simultaneously with matrices more like a Direct Conversion Algorithm or Flash Converter (see Section 2.3.1).

4.2.1 Thresholds of decision

The first things that have to be known are the thresholds for the conversion, the values that distinguish an output value of another. These thresholds shall be stored in a vector whose first position corresponds to the threshold of the first change of the output (from the digital value "0" to the digital value "1") and so on. It should be noted that, as it will be seen later and because of the implementation of the converter, the last value of the thresholds is special because it does not affect to the calculation of non-idealities because it is referred to the change of the output of digital value between $(2^{\text{bits number}} - 1)$ and $2^{\text{bits number}}$, that it is outside of the operating range of the output of the converter (between the digital values of 0 and $2^{\text{bits number}} - 1$).

These thresholds are generated by combining different displacement values of the thresholds of the ideal case and the slopes of the transfer function between two consecutive thresholds.

Ideal case is defined as the converter whose output value increases linearly with each multiple value of LSB [6].

```
% ADC's ideal transfer function
idealConverter = (1:2^nbits)*vlsb;
```

Figure 4.1: *Extract from ADC.m that generates an ideal converter*

Example 4.1

Generate an ideal converter.

Solution:

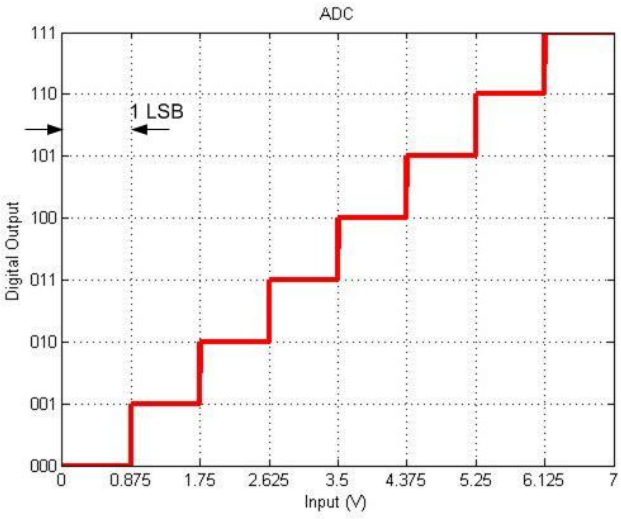


Figure 4.2: *3-Bit Ideal Converter,* $V_{REF} = 7$

As it can be appreciated, the LSB value is $\mathrm{V_{REF}}/2^{\mathrm{n}}$, where $\mathrm{V_{REF}}$ is the value of the reference voltage applied to the converter (which multiplies the input signal and represents the maximum possible value of the output voltage) and n is the number of bits with which the conversion is performed. This value will be known henceforth as $\mathrm{V_{LSB}}$.

```
% Vlsb value
vlsb = vref/2^nbits;
```

Figure 4.3: *Extract from ADC.m that calculates* V_{LSB}

To higher n, higher resolution, so they can capture smaller voltage changes, albeit at a higher cost, and with the problem of the presence of noise. To lower V_{REF} also will have a lower dynamic range, suffering again problems with noise and therefore the SNR performance.

4.2.1.1 Parameter "steps"

When running the program, as can be seen in [*Annex I*] it can be defined parameter "*steps*". To use this parameter two different options are possible.

One would be to use a single value that would move to all the thresholds of the ideal case as much V_{LSB} as it indicates. For the case of Figure 4.2 the thresholds are at every multiple of V_{LSB}, if for example it is entered a value of "*steps*" of -0.5, all thresholds would move $\frac{1}{2}V_{LSB}$ to the left, resulting:

Example 4.2

Generate a converter with $-0.5V_{LSB}$ of offset.

Solution:

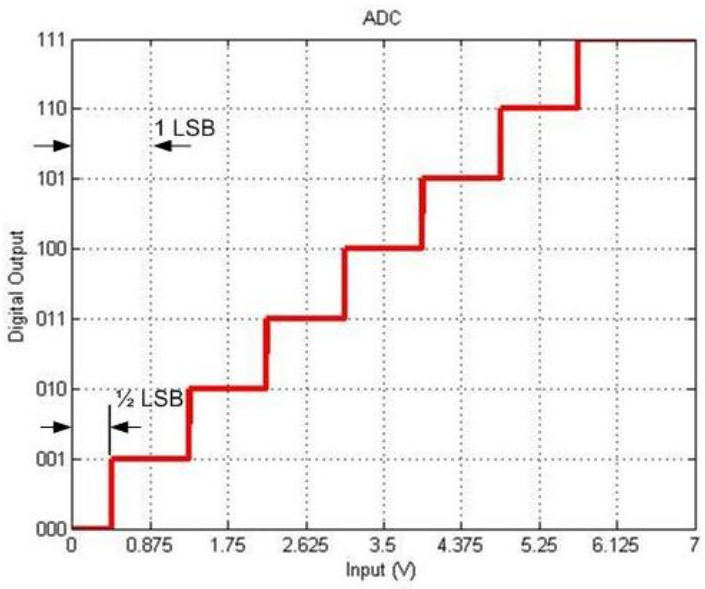


Figure 4.4: *3-Bit Converter with* $steps = -0.5$, $V_{REF} = 7$

As it can be guessed, the offset value accepts both positive and negative values, and will always be in units of $\mathrm{V_{LSB}}$.

The other option would be to use a vector with as many values as thresholds that has the converter (i.e., 2^{n}), where each threshold of the ideal case will move according to the corresponding value. It has to keep in mind that the first vector value corresponds to the threshold located in $1\mathrm{V_{LSB}}$, or what it is the same, $\mathrm{V_{REF}}/2^{\mathrm{n}}$, which corresponds to the digital leap from "0" to "1", and the last value of the vector to the $\mathrm{V_{REF}}$ threshold. But as it was stated above, this latter value does not affect the calculation of the non-idealities, so it is recommended to leave it to 0 and if it is wanted to modify the step corresponding to the digital output "$2^{\mathrm{n}} - 1$", it should be modified the penultimate threshold.

Example 4.3

Generate a converter with parameter $steps = [0\ -0.7\ 0\ -0.5\ 0.5\ 0\ 0.7\ 0]$.

Solution:

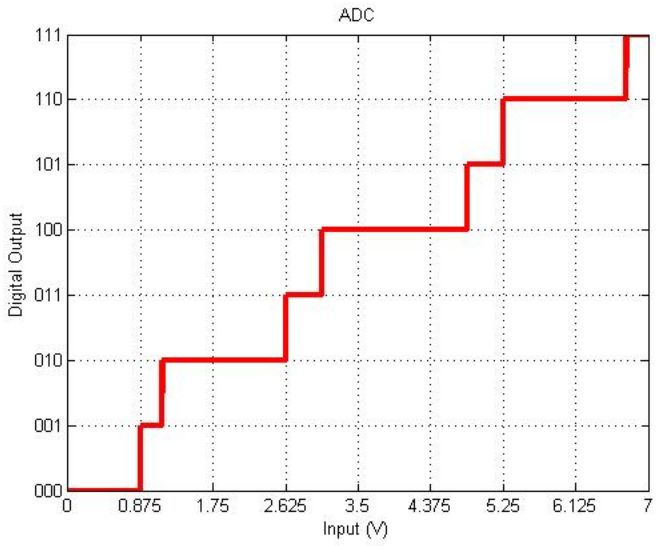
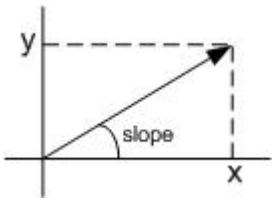


Figure 4.5: *3-Bit Converter with* $steps = [0\ -0.7\ 0\ -0.5\ 0.5\ 0\ 0.7\ 0]$, $V_{REF} = 7$

4.2.1.2 Parameter "slopes"

The second parameter that can be modified is "*slopes*", that defines the slope of the transfer function. In the ideal case the slope is 1, this means that each increase in a multiple of $\mathrm{V_{LSB}}$ of the input voltage supposes increasing by one unit the digital output value.



$$slope = \frac{y}{x}$$

Figure 4.6: *Definition of slope*

If for example the slope was 2, then for each increment of V_{LSB} at the entrance it will be an increase of two units in the output (the output will increase one unit in each $\frac{1}{2}\mathrm{V_{LSB}}$).

Example 4.4

Generate a converter with $2V_{LSB}$ of slope.

Solution:

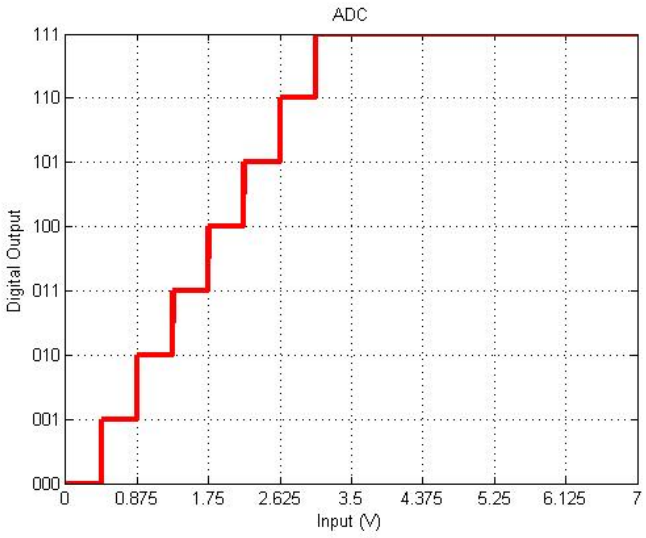


Figure 4.7: *3-Bit Converter with slopes* $= 2$, $V_{REF} = 7$

On the contrary, if the value was 0.5, it is needed an increase of $2V_{\mathrm{LSB}}$ to make a change in the value of the output signal.

Example 4.5

Generate a converter with $0.5V_{LSB}$ of slope.

Solution:

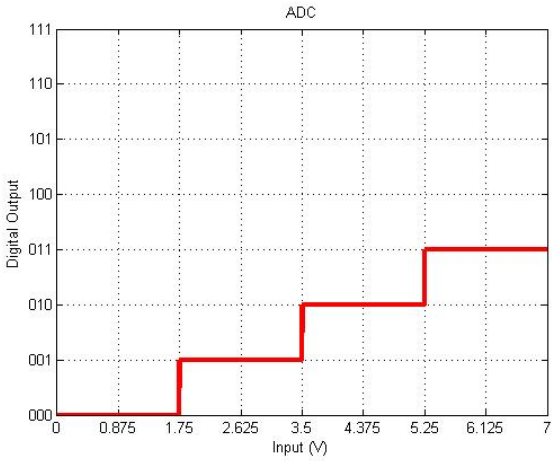


Figure 4.8: *3-Bit Converter with* $slopes = 0.5$, $V_{REF} = 7$

Obviously the value of "*slopes*" must be greater than 0, and, as in the case of the parameter "*steps*", it can be used a vector of 2^{n} values that represents the slope of each one of the intervals or steps. It must be also taken into account the peculiarity of the last value of the vector, so in a similar way as it happens in the case of "*steps*", it is recommended to leave this last value to 1.

Example 4.6

Generate a converter with parameter $slopes = [1.2\ 1\ 1.5\ 1\ 1\ 0.5\ 1\ 1]$.

Solution:

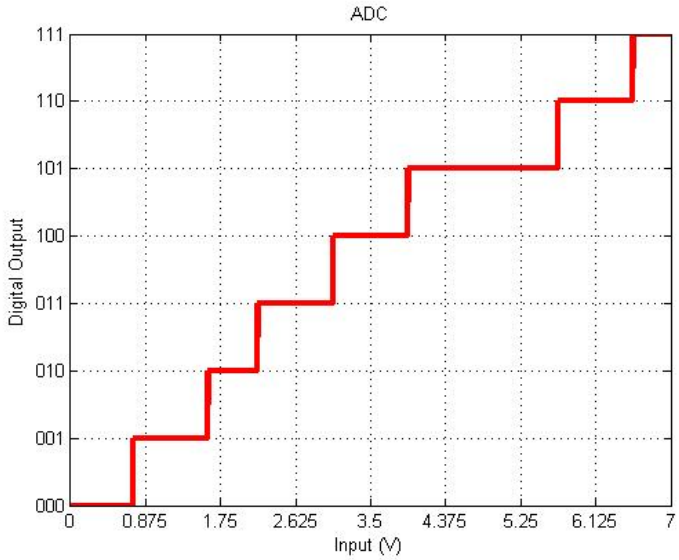


Figure 4.9: *3-Bit Converter with* $slopes = [1.2\ 1\ 1.5\ 1\ 1\ 0.5\ 1\ 1]$, $V_{REF} = 7$

Combining these two parameters it is calculated the position that would correspond to each new threshold moving it from its position in the ideal case. In this step is important to note the manner in which the displacement is calculated for each threshold because of the slope when it is being using a vector instead of a single value, thus for a correct calculation for each threshold, it should be taken into account the movement of all previous thresholds (remembering that all calculations are performed in $\mathrm{V_{LSB}}$ units) because the slope value only refers to an interval, regardless where it starts.

$$slopes = [s_1 \quad s_2 \quad s_3 \quad \cdots \quad s_N], N = 2^n \tag{4.1}$$

$$slope\_steps = \left(\frac{1}{s_1} \quad \frac{1}{s_2} \quad \frac{1}{s_3} \quad \cdots \quad \frac{1}{s_N}\right) * \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}_{N \times N}, N = 2^n \tag{4.2}$$

The thresholds will be stored in a matrix of 2 rows in which each of the thresholds is associated to the digital output value for lower inputs than that threshold, and larger than the previous threshold.

```
% ADC's real transfer function
converter = zeros(2,2^nbits);
% Thresholds of conversion
slope_steps = ((ones(1,2^nbits)./slopes)-
1)*triu(ones(2^nbits,2^nbits));
converter(1,:) = ((1:2^nbits)+slope_steps+steps)*vlsb;
% Digital output values
converter(2,:) = (1:2^nbits)-1;
```

Figure 4.10: *Extract from ADC.m that calculates the thresholds*

4.2.2 Comparison matrices

Once known the thresholds used for the data conversion, it must be obtained the matrices that will be used for calculations.

$$thresholds = [u_1 \quad u_2 \quad u_3 \quad \cdots \quad u_N], N = 2^n \tag{4.3}$$

In the code, these thresholds are stored in "*converter (1,:)*" [*Annex I*]

First it is built the reference matrix, known as "*aux*". That matrix contains in each of the columns (as many as the length of the input signal) the squared value of the conversion thresholds. I.e.:

$$aux = \begin{pmatrix} u_1^2 & u_1^2 & u_1^2 & \cdots & u_1^2 \\ u_2^2 & u_2^2 & u_2^2 & \cdots & u_2^2 \\ u_3^2 & u_3^2 & u_3^2 & \cdots & u_3^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_N^2 & u_N^2 & u_N^2 & \cdots & u_N^2 \end{pmatrix}_{N \times M}, N = 2^n, M = input\ length \tag{4.4}$$

After, it is generated the matrix that carries the information of the input values, that will be named as "*conversion*" and it will be compared with the reference matrix. The input values have to be first adjusted to facilitate the comparison, because the code is adapted to entries with values adjusted to the range [0,2] although this can be easily modified in the line where it is weighted up the input signal and the reference voltage.

What is finally obtained, is a matrix with as many columns as the length of the input signal, and with 2^n rows. Each column corresponds to an input value, and all the values of a same column are the products of the value of the normalized input signal (the highest possible input value,

in this case, as mentioned in the previous paragraph, 2, it is normalized to V_{REF}) for each one of the thresholds:

$$conversion = thresholds * input = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{pmatrix} * (v_1 \quad v_2 \quad v_3 \quad \cdots \quad v_M) =$$

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1M} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2M} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & w_{11} & w_{NM} \end{pmatrix}, N = 2^n, M = input\ length \tag{4.5}$$

```
% Auxiliary (2^nbits)x(length(vin)) matrix with the square of each digital
% value in the columns
aux = vin*0+1;
% Weight up the new matrix with the reference voltage and the number of
% bits
aux = (converter(1,:)'.^2*aux);
% Weight up the input and compare it with the auxiliary matrix
conversion = converter(1,:)'*(vin*vref/2);
```

Figure 4.11: *Extract from ADC.m that generates the comparison matrices*

4.2.3 Calculation of the digital output

To calculate the digital output it just have to compare the obtained matrices in (4.4) and (4.5) position by position, having done that, two options have been studied, both based mainly on how to approach the problems of the Negative Thresholds because of offset (see Section 4.3.1.1) and the Bubbles (see Section 4.3.1.2).

The implemented solution in the final version, instead of adding the number of rows at "1", fixed just on the number of the row that contains the last "1" (when made the comparison, if it is met that the value of the input is greater than the reference in that position, in the matrix with the solution it is obtained a "1" in that position, otherwise the solution matrix has a "0") in ascending order. This solves the problem of the Bubbles, but it presents the problem of Negative Thresholds in the lowest value of the digital output which will be solved later.

```
conversion = conversion>aux;
% After fixing the problem of bubbles the output will be the number of ones
% in each column
out = max(((1:2^nbits)'*ones(1,length(conversion))).*conversion);
```

Figure 4.12: *Extract from ADC.m that calculates the output*

In an alternative version it has been developed a more complex code for the location and the removal of Bubbles (see Section 4.3.1.2 and [*Annex IV*]) where the result for the digital conversion for each input value is the number of rows where the matrix with the entry information ("*conversion*") is greater than the reference ("*aux*") in the corresponding column.

By itself, this alternative version does not solve the problem of the minimum output value when there are Negative Thresholds, but it offers another approach when calculating the output.

Finally to stress the cases where there is a maximum of the digital output. Because of the way in which the code is implemented, this occurs when all rows of one column in the comparison matrix are "1" (in case of Bubbles and Negative Thresholds, that does not happen, but despite that, what matters is that the last row is worth "1") so when this output is generated, the value will be 2^{n}. Given that the maximum digital output is $2^{\mathrm{n}} - 1$, these values should be limited.

```
% Limit maximum output value
out(out>2^nbits-1) = 2^nbits-1;
```

Figure 4.13: *Extract from ADC.m that limits the output*

4.2.4 Information about non-idealities

As supplementary information, the program calculates a set of non-idealities of the generated converter. More specifically it provides the values of Offset Error, Full-Scale Error, Gain Error, DNL and INL (see Sections 3.2.5 and 3.2.6).

All the values are expressed in units of $\mathrm{V_{LSB}}$ and are calculated respect the best linear approximation of the values of the thresholds of the converter.

More specifically, the Offset Error value is the value of the approximation for the digital output "0". The Full-Scale Error is the difference between the value of the approximation and the value of ideal converter for the digital output "$2^n - 1$". The Gain Error is the Full-Scale Error after removing the Offset Error. DNL is the maximum difference between the widths of each step of the transfer function and $1V_{LSB}$ (the program returns a vector with the size of each

step in V_{LSB}, to know the value of DNL it would be necessary execute "*max(abs(dnlvector))*"). Finally, INL is the maximum difference between the input value for a given output of the real case, and the input value for the same output in the best approximation (as in the DNL case, the program returns a vector with all the differences and to get the value it just have to execute "*max(inlvector)*").

However it should be pointed the fact that to obtain the linear approximation it has been used the *polyfit* function that provides MATLAB. This function has the particularity that due to the features of MATLAB the adjustment is not 100% accurate, as the program itself makes some approximations, which although not generate a large error (around 10^{-16} or even less), it is striking.

```
% ADC's transfer function's linear interpolation
approxFunction = polyfit([steps(1) converter(1,1:end-
1)/vlsb],converter(2,:),1);
approxConverter = (converter(2,:)-
approxFunction(2))*vlsb/approxFunction(1);
```

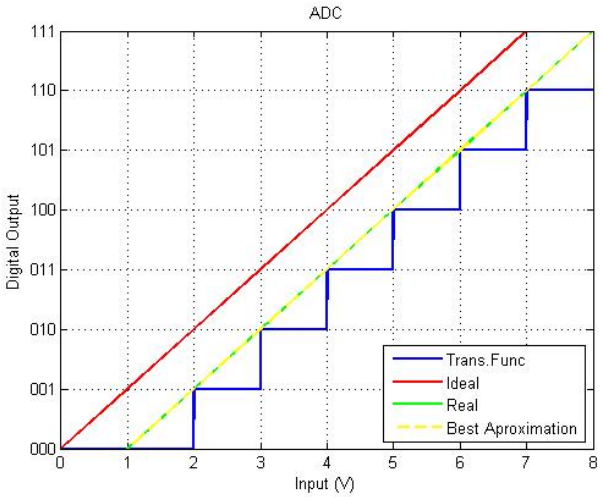
Figure 4.14: *Extract from ADC.m that generates the best linear approximation of the converter*

To represent this fact it is showed the next execution of the program where there is used a converter with the value $\text{steps} = 1$ and the obtained values of non-idealities:

Example 4.7

Generate a converter with $1V_{LSB}$ of offset and see the non-idealities.

Solution:



Non-Ideality	Value
Offset Error	1.0000
Gain Error	$2.2204 * 10^{-16}$
Full-Scale Error	-1
DNL	1
INL	$2.2204 * 10^{-16}$

Figure 4.15: *3-Bit Converter with steps* $= 1$, $V_{REF} = 8$, *and non-idealities*

Theoretically, Gain Error and INL should be 0 exactly, functionally they are, because a so small error does not affect to the output. The error comes from the mismatch of the Offset Error value, which should be 1, but as it can be supposed by the presence of the decimals (although initially are 0) the value, although in a very small scale, does not conform to the theoretical value.

Obviously this characteristic is only appreciable in specific cases and simple as this one, in other cases this fact of MATLAB goes unnoticed.

Furthermore, it must pointed out that unlike the real converter, which is defined by the threshold values starting at the change from "0" to "1", and therefore it begins to be defined at "1", the approximation, as it needs an initial value for the output "0" (among other reasons, to be plotted a posteriori), starts at 0. Therefore for the calculation of non-linearity it is necessary to adjust the first and last values to compare the two converters.

```
% Output info
offset = approxConverter(1)/vlsb;
FSError = (idealConverter(end-1)-approxConverter(end))/vlsb;
GError = FSError+offset;
aux_inl = converter(1,1)-vlsb/slopes(1);    % INL Adjustment
INL = abs(approxConverter-[aux_inl converter(1,1:end-1)])/vlsb;
DNL = ([converter(1,1:end-1) vref]-[0 converter(1,1:end-1)])/vlsb-
1;
DNL(DNL<=-1) = 0;   % DNL Adjustment
```

Figure 4.16: *Extract from ADC.m that calculates the non-idealities*

It has to be pointed out the process of calculating the value of INL. As the implementation of the code is done for that the variable *converter (1,:)* contains the thresholds (where happens the change of the output codes) starting with the threshold between the codes "0" and "1", to calculating the value of the deviation in the "0" is necessary to calculate by separate (variable *aux_inl*) with the values of the code "1" and the slope of the converter in that interval.

In addition, to ensure that the calculations of DNL are correct it is necessary to ensure that no values are lower or equal to -1, which would mean that there are Missing Codes, since them could distort the results of the maximum value.

4.2.5 Graphical Information

To make the study and analysis of the obtained results easier, it has been implemented a function that generates the graphical representation of the converter and the relationship between the calculated signals of input and output [*Annex II* and *Annex III*].

Note that this graphical representation is limited to the operating range of the converter (between 0 and V_{REF}).

The program lets to choose between representing only the transfer function of the converter, or if on the contrary, it is preferred to represent all available information (transfer function, threshold values for the ideal cases, real and best linear approximation, and comparison between the analog input and the digital output).

```
% Plotting
if strcmp(plotting,'full')
      ADCplotting(vin,out,nbits,vref,idealConverter,converter,appro
      xConverter,vlsb,slopes(1));
elseif strcmp(plotting,'simple')
    ADCplottingSimple(nbits,vref,converter,vlsb);
else disp('Wrong plotting input value');
end
```

Figure 4.17: *Extract from ADC.m for the different plotting options*

Two options were proposed to represent the transfer function of the converter, the one which has finally been implemented consists on setting the number of points that are represented regardless of the value of V_{REF}, and go overlapping the values of the vector that contains the data to represent, with the different digital output values in ascending order and according to the threshold specified by the converter.

```
% Number of points in the plot
voltPoints = 2^14;
converterRep = zeros(1,voltPoints);

------------------------------------------------------------

% Representation of the ADC transfer function
scaling = voltPoints/vref;
scaledConverter = converter(1,:)*scaling;
scaledConverter(scaledConverter<0) = 0;
for i = 2:2^nbits+1
    converterRep(round(scaledConverter(i-1))+1:end) = i-1;
end
converterRep(converterRep>2^nbits-1) = 2^nbits-1;
```

Figure 4.18: *Extracts from ADCplotting.m and ADCplottingSimple.m that generates the ADC transfer function*

This option has the advantage that it is much simpler than the alternative (see Section 4.3.2), requiring less adjustments, conditions and parameters. By contrast the other version generates the number of points of representation depending on V_{REF} (filling only the corresponding interval of the step), which requires to take care about very small values, but as it is scaled, it allows that each point on the graph corresponds to a "real" input voltage (for example the 3.125 value would be in position 3125). However, as it was mentioned above, it has been finally opted for the simple version because if it is wanted to know the exact voltage value, it is enough with multiply the variable "*scaling*" and the position.

Example 4.8

Generate a converter and see the full graphical information.

Solution:

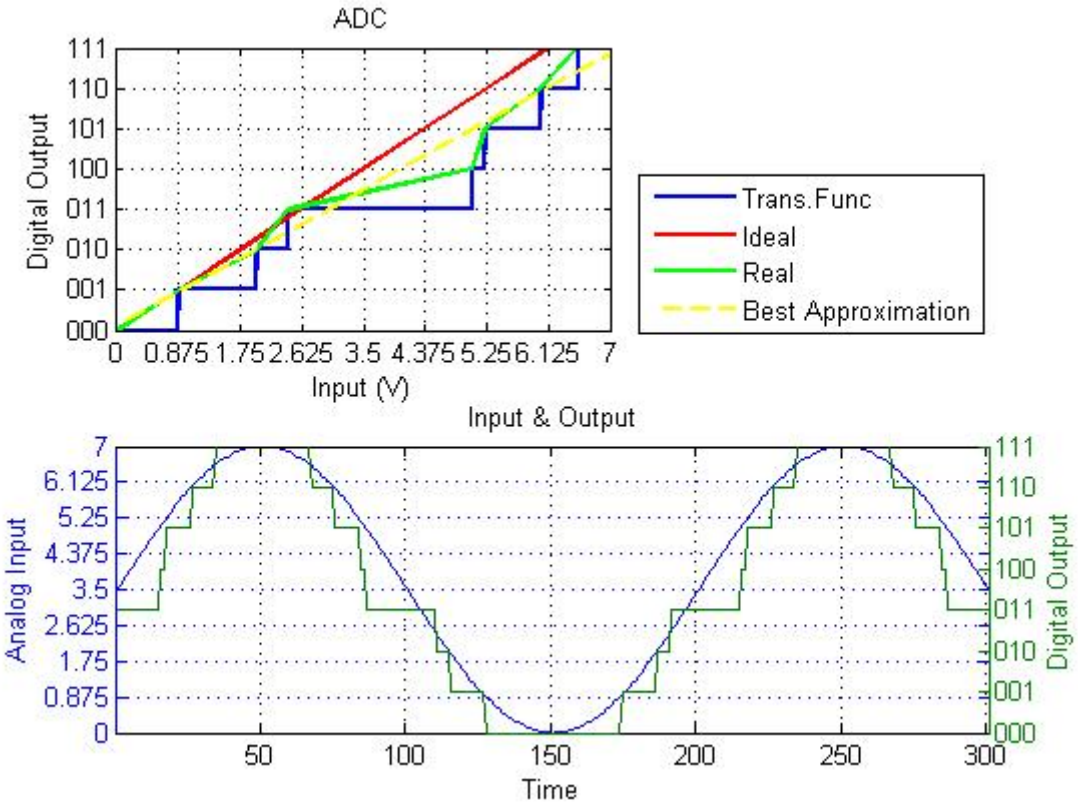


Figure 4.19: *3-Bit Example Converter,* $V_{REF} = 7$

4.3 PROBLEMS FOUND AND ALTERNATIVE SOLUTIONS

Throughout the development of this project, there has been found some difficulties and problems to solve, to achieve proper operation of the converter. When resolving these problems, several options arose and have been chosen those that provide a greater simplicity, clarity and efficiency. However, the other options might be useful in some other situation, so, those problems and the discarded solutions will be explained then.

4.3.1 Problems when calculating the value of the digital output

Under normal conditions, the resulting matrix to compare the reference matrix with the one that contains the input information will be a binary matrix with between 0 and 2^n "*ones*" in each column, starting at the first position and ending on the position that corresponds to the output value. For example:

Example 4.9

Given the matrix $\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$, what digital output corresponds?

Solution:

$$(4 \quad 8 \quad 2 \quad 6 \quad 0 \quad 4) \xrightarrow{limiting} (4 \quad 7 \quad 2 \quad 6 \quad 0 \quad 4) \tag{4.6}$$

However there are situations where this is not true due to the implementation done.

4.3.1.1 Negative Thresholds

With the term Negative Thresholds is pretended to appoint the situation when, for any reason, it is have that the value of one or more of the thresholds (typically the lowest ones) is negative or "0" (that would cause the lowest codes to become Missing Codes), the matrix resulting from the comparison will submit all 0s in its first R rows (where R is the highest position of a null or Negative Threshold and taking into account that the thresholds are in ascending order with respect to the ideal case, and that a threshold corresponding to $2V_{LSB}$ can have a value lower than $1V_{LSB}$).

Example 4.10

Given several vectors of thresholds, what is the R-value of each?

Solution:

$$[0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7] \rightarrow \mathrm{R} = 1 \tag{4.7}$$

$$[0 \quad -0.3 \quad 1 \quad 3 \quad 3.5 \quad 5 \quad 4 \quad 7] \rightarrow \mathrm{R} = 2 \tag{4.8}$$

$$[0.4 \quad 0.1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7] \rightarrow \mathrm{R} = 0 \tag{4.9}$$

$$[0.5 \quad 1 \quad 1.5 \quad 2] \rightarrow \mathrm{R} = 0 \tag{4.10}$$

$$[1 \quad 1.7 \quad 0 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7] \rightarrow \mathrm{R} = 3 \tag{4.11}$$

In the implemented solution it has been decided to set all thresholds below the one which is located at position R to 0 to avoid problems with calculations.

```
% Readjust negative thresholds
R = find(converter(1,:)<=0,1,'last');
converter(1,1:R)=0;
```

Figure 4.20: *Extract from ADC.m that fixes the thresholds*

Example 4.11

How would be the matrix of Example 4.9 with a value of R = 2? What output would be generated?

Solution:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{adding} (2 \quad 6 \quad 0 \quad 4 \quad 0 \quad 2) \tag{4.12}$$

However, the solution of (4.12) is incorrect because all values of the first R rows should be "1" (as it have Negative Thresholds or lower than 0, the minimum digital output will be R), but due

to the implementation of the matrices this is not satisfied, therefore it is necessary to readjust the value a posteriori.

This problem really only affects the outputs with the minimum digital value (those columns that present no "1") because in either of the solutions developed for calculating the digital output (see Section 4.2.3) when a column has at least a position with a "1" all previous positions are assumed (with the implemented case where only takes into account the higher position to "1") or converted (with the search algorithm of Bubbles (see Section 4.3.1.2)) into "1".

Therefore to solve the problem it is enough with readjust the outputs whose value is "0".

```
% Compensation of the 0s for non-positive thresholds
if R>0
    out(out==0) = R;
end
```

Figure 4.21: *Extract from ADC.m that compensates the 0s*

4.3.1.2 Bubbles

The problem of the Bubbles [16] is interesting because it is a problem that also happens in real converters. Being necessary to implement a logic block, which it solves the problem.

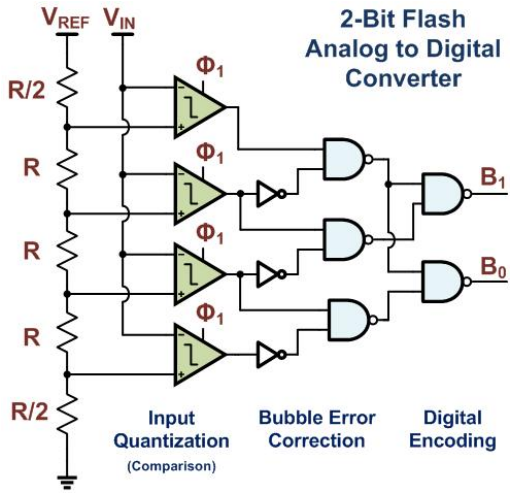


Figure 4.22: *2-Bit Flash ADC with Bubble corrector*

It shows a great similarity with the problem of Negative Thresholds because it consists in the emergence of 0s in the positions that should have 1s.

The cause of the Bubbles are the Missing Codes, where a higher threshold overlaps (is smaller) to a lower one, basically the same as in the Negative Threshold, except because of the difference that this phenomenon can occurs in any of the codes and does not affect to all the values of the previous rows, but only to the row of the Missing Code in the columns where the entry has a value within the range of the overlapped thresholds.

Below there is an example of this situation together with the detail of the values of the comparison matrix at the positions where the entrance is around the range of overlapping:

Example 4.12

What happens to the comparison matrix when thresholds are overlapping?

Solution:

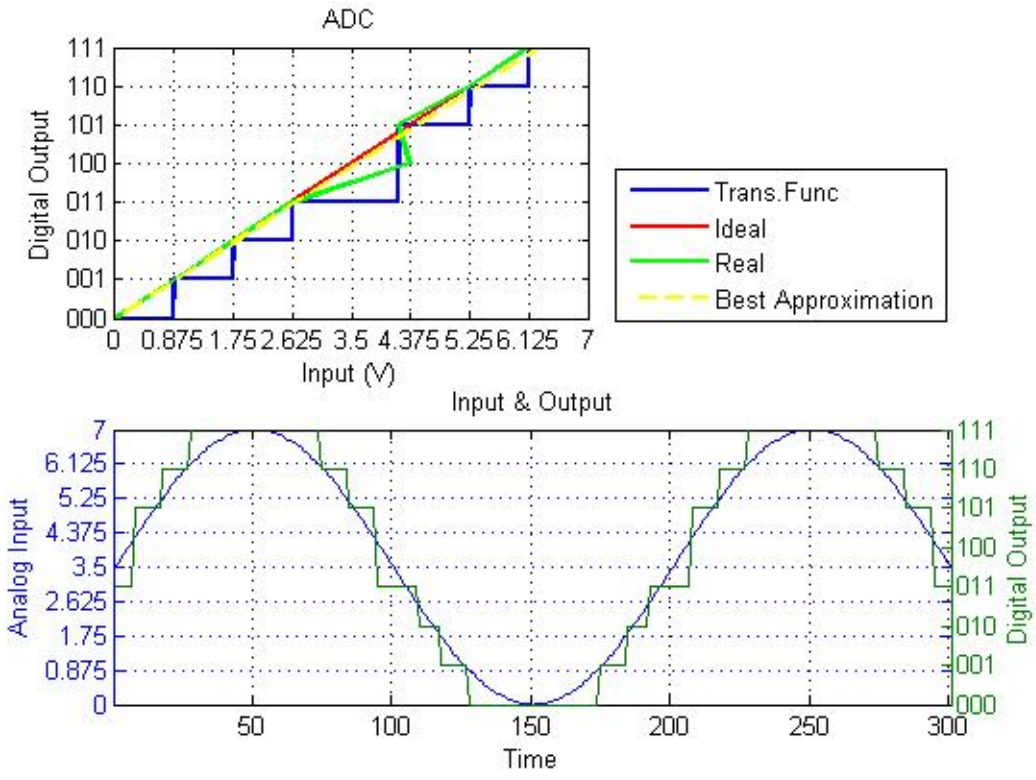


Figure 4.23: *3-Bit Converter with Missing Code,* $V_{REF} = 7$

Range of overlapping: [4.2,4.375]

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{adding} (5 \quad 5 \quad 4 \quad 4 \quad 3 \quad 3) \tag{4.13}$$

Obviously, in (4.13) the code 4 ('010' in 3-bit binary) should not appear at the output since it is missing. With the solution used in the practice of only take into account the position of the last row that is at "1", this problem does not affect the operation, but alternatively it has been developed a code [*Annex IV*] that finds the positions of the Bubbles (due to both Missing Codes and Negative Thresholds) and change their values from "0" to "1".

This code is designed for analog-digital converters, so it only will work for matrices whose number of rows is a power of 2, although it can be adjusted to matrices of any size by adding an entry that would be the number of rows, and modifying minimally the code. The operation consists in generating a matrix where each row corresponds ascending to a power of 2, and that only presents non-zero values at positions with "1" of the matrix that is being analyzed. Adding all values in the same column will give a value that has to be compared with the maximum possible value that would occur in this column if it had no bubbles (the sum of all powers of 2 from 0 to the last row to "1"). The positions with Bubbles are the "*ones*" of the difference between the two values in binary.

The code returns the position of the matrix where the Bubbles are, but not in the nomenclature (*row*,*column*), but as a single number. Should be remembered, that in MATLAB, the positions of a matrix are assigned in ascending order by columns: in a $N \times M$ matrix, position (1,1) has the value 1, the (N,1) is N, the (1,2) is N+1, the (N, 2) is 2N and so on.

Example 4.13

Given the matrix $\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ find the Bubbles.

Solution:

$$\begin{pmatrix} 2^0 & 2^0 & 2^0 & 2^0 \\ 2^1 & 2^1 & 2^1 & 2^1 \\ 2^2 & 2^2 & 2^2 & 2^2 \\ 2^3 & 2^3 & 2^3 & 2^3 \end{pmatrix} .* \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2^0 & 2^0 & 0 & 2^0 \\ 2^1 & 0 & 2^1 & 2^1 \\ 0 & 2^2 & 2^2 & 0 \\ 0 & 0 & 2^3 & 2^3 \end{pmatrix} \rightarrow \begin{Bmatrix} sum = (3 & 5 & 14 & 11) \\ max = (3 & 7 & 15 & 15) \end{Bmatrix}$$

$$\xrightarrow{subtracting} (0 \quad 2 \quad 1 \quad 4) \xrightarrow{to\ binary} ('000' \quad '010' \quad '001' \quad '100')$$

$$\rightarrow Bubbles\ in\ (2,2), (1,3) and\ (3,4) \rightarrow Bubbles\ in\ 6, 9\ and\ 15 \qquad (4.14)$$

4.3.2 Representation of the transfer function of the converter

As it was already pointed out above, to generate the representation of the transfer function of the converter two options were handled.

Initially the solution suggested was to generate the output values range by range, but this caused many mismatches that were necessary to remedy, and a much more complex code [*Annex V*]. Finally it was decided that the implemented solution was the previously discussed on paragraph 4.2.5.

5 EXAMPLES

The purpose of this chapter is to review the concepts explained throughout the project on the basis of the results obtained with the program developed in MATLAB. For this, it will be made some executions and the results will be displayed, progressing gradually to show the different striking aspects, serving also as an aid to understanding some of the parameters explained in Chapter 2.

To start it is generated an ideal ADC:

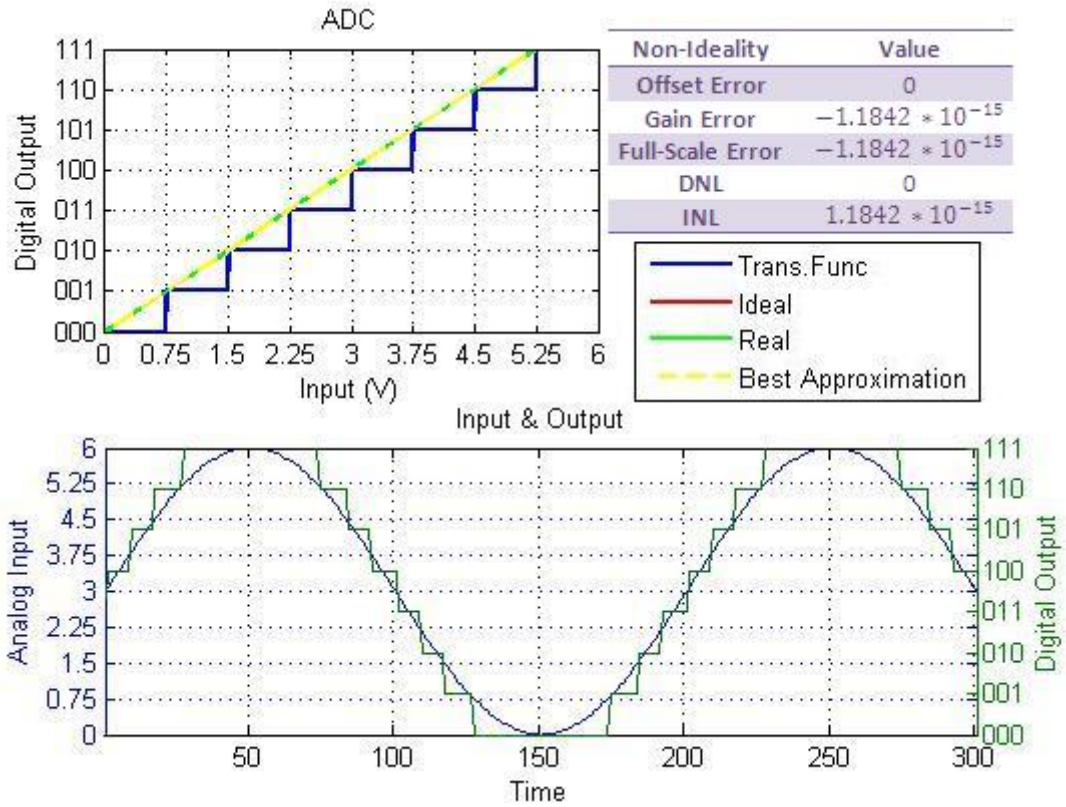


Figure 5.1: *Ideal 3-Bit Converter,* $V_{REF} = 6$

It was already discussed in Section 4.2.4 the fact of the approximations of the *polyfit* function of MATLAB, in practice all the values of non-ideality will have in this case a value of 0. Also, it can be appreciated in the graph comparing the input and output, as all jumps correspond to the appropriate multiple of LSB.

Now it will be generated a converter with Offset Error and Full-Scale Error, for it, simply it has to be changed the input parameters *steps* and *slopes*.

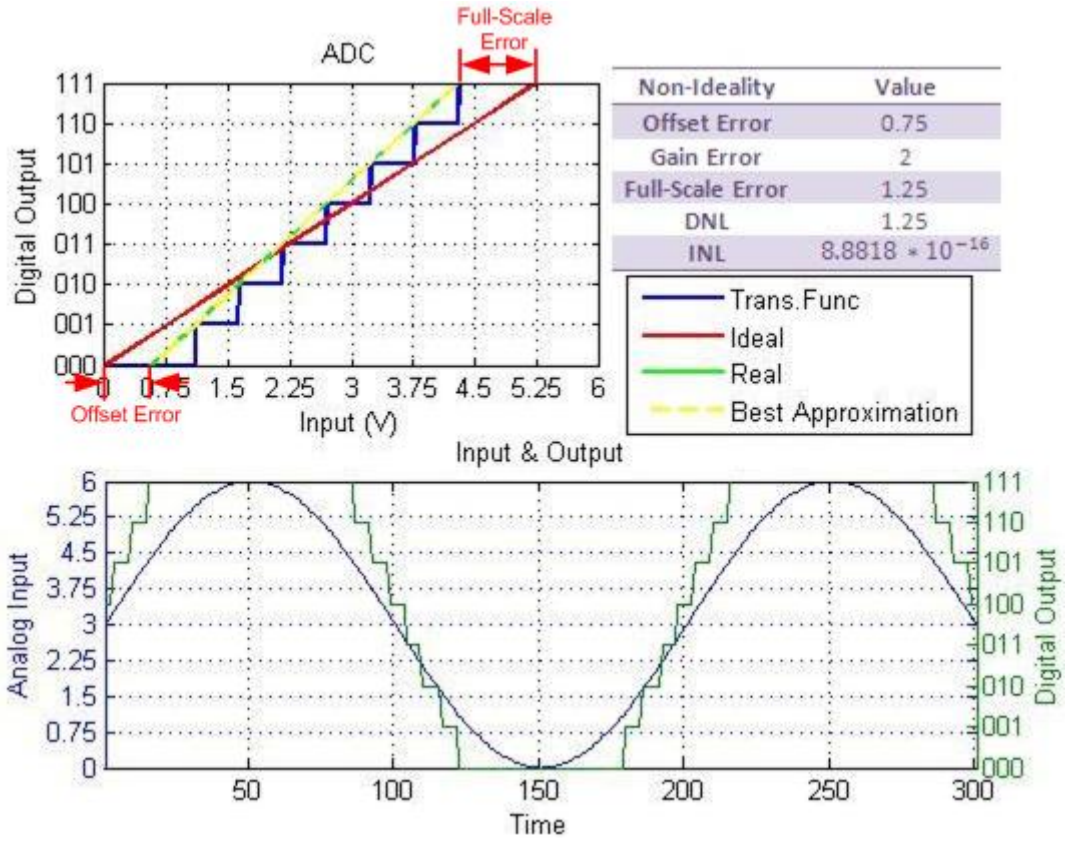


Figure 5.2: *3-Bit Converter with* $steps = 0.75$ *and* $slopes = 1.4$, $V_{REF} = 6$

Here, there can be seen the first two important concepts, Offset and Full-Scale errors. It should be remembered that the value of these errors is expressed in units of LSB, while the scale of the graph of the transfer function is in natural units of voltage, in this case the relationship will be:

$$LSB = \frac{V_{REF}[V]}{2^{nbits}} = \frac{6}{8} = 0.75[V] \tag{5.1}$$

It can be also appreciated now that the steps for the digital outputs '000 'and '111' are larger than in the case of Figure 5.1, which confirms the fact that the output wave has a greater number of values that take these digital codes at the output than in the previous case. This will also affect the value of DNL, but this concept will be explained later. It should be note further that the other codes have been displaced with respect to multiples of LSB.

It was already explained that to obtain the value of the Gain Error, it has to be make zero the Offset Error. Knowing that for the previous case the Offset Error is 0.75, it is enough to make null the parameter *steps* and to maintain *slopes*, and as the Offset has been "eliminated" from the previous example, the value of Gain Error must match the value of the table Figure 5.2:

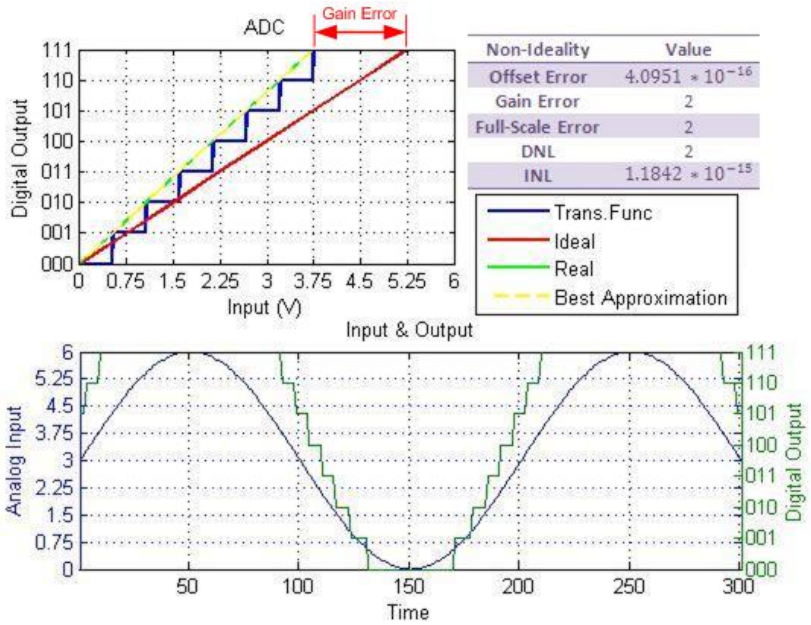


Figure 5.3: *3-Bit Converter with* $steps = 0$ *and* $slopes = 1.4$, $V_{REF} = 6$

While the value of *steps* used in this case is 0, because of the approximation of the function *polyfit*, it indicates that Offset Error is not exactly 0, but as it can be seen, functionally it is, since Gain Error corresponds to the case in Figure 5.2 (this is not the cause of the Full-Scale Error, which coincides with the Gain Error because in this example there is no Offset).

As it already can be appreciated, the implementation of the code causes that the positive values of Offset are considered to the right of the ideal case, while for the Full-Scale and Gain Errors, positive values are taken to the left of the ideal case:

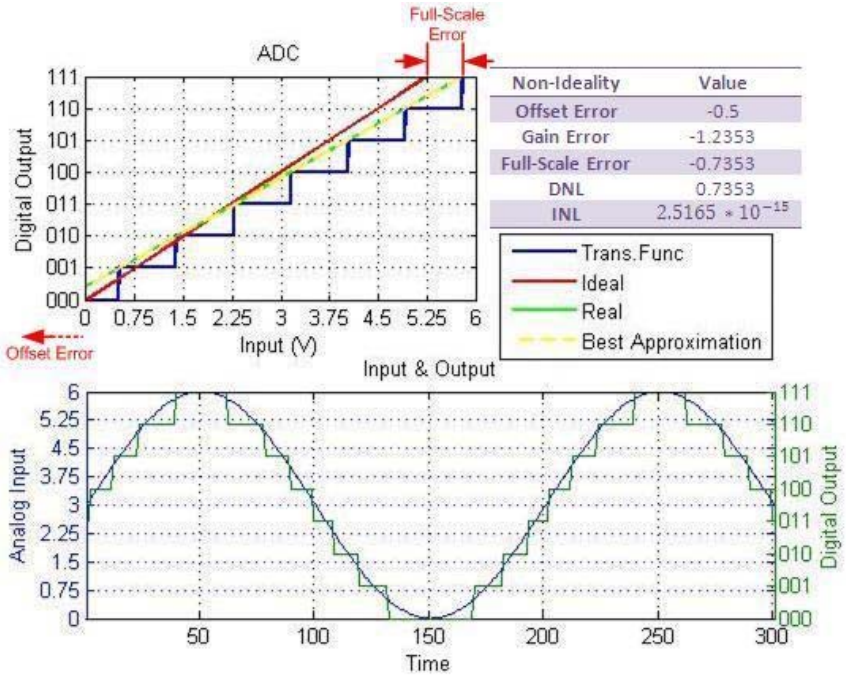


Figure 5.4: *3-Bit Converter with* $steps = -0.5$ *and* $slopes = 0.85$, $V_{REF} = 6$

In this case, canceling the Offset Error to get the Gain Error, the line of the best linear approximation will shift to the right, so that Gain Error will be smaller (larger negative value) than the Full-Scale Error.

Returning to the concept of DNL, it can also be analyze the value of INL. The DNL parameter represents the greatest deviation of the width of a step of the converter with regard the ideal case of $1LSB$, both positively (steps wider than $1LSB$) and negative (steps smaller than $1LSB$). Until now, it has been seen as all values of INL are essentially 0, as the generated converters have been linear; but playing with the parameters *steps* and *slopes* it is reached that the converter ceases of being linear, from here, it must be performed the approximation that best fits (using *polyfit* as previously discussed). The value of INL will be the largest deviation between the approximation and the real values of the thresholds of the converter for each code.

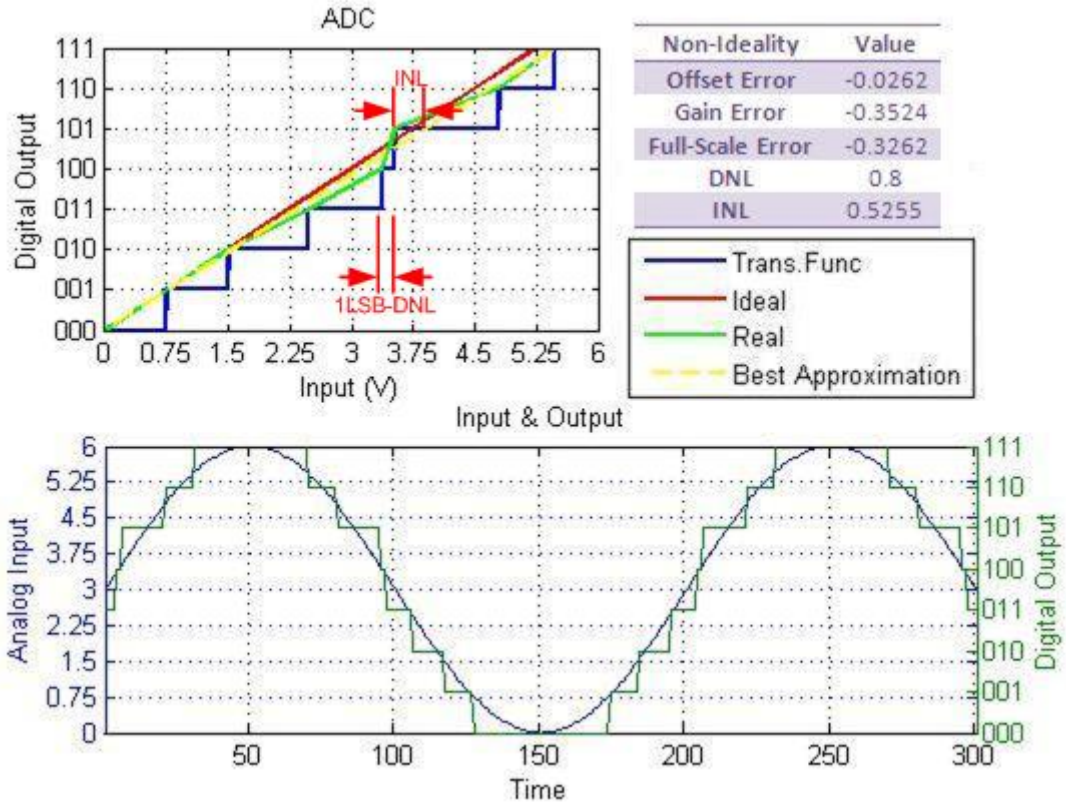


Figure 5.5: *3-Bit Converter with steps* $= [0\ 0\ 0.3\ 0.5\ -0.3\ 0.4\ 0.3\ 0]$, $V_{REF} = 6$

It should be remembered that the program returns vectors for both, the DNL and INL, to facilitate the locating of the corresponding value. To obtain the exact value it is enough with calculate the maximum (for the case of DNL it will have to seek the maximum in absolute

value). In this case the step corresponds to a DNL less than $1LSB$, but also it may correspond to a bigger one:

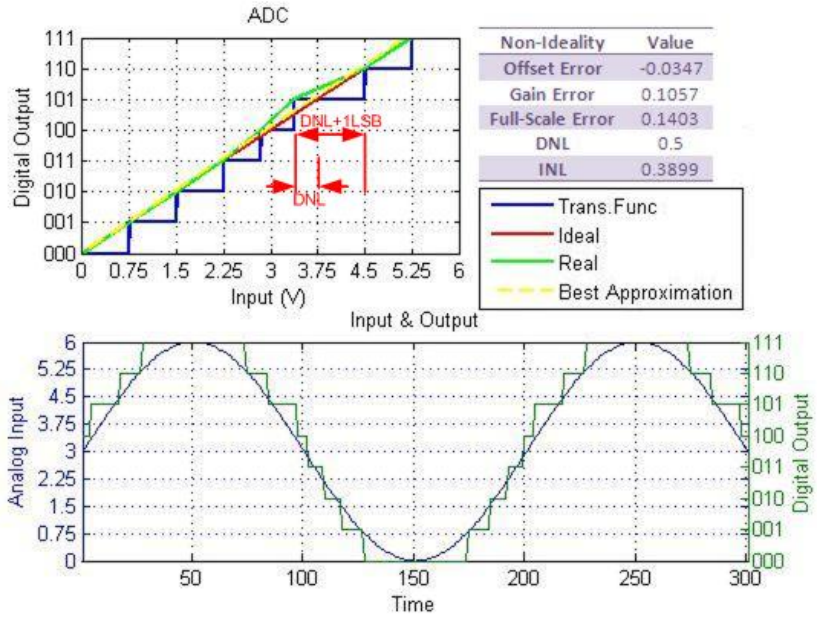


Figure 5.6: *3-Bit Converter with steps* $= [0\ 0\ 0\ -0.2\ -0.5\ 0\ 0\ 0]$, $V_{REF} = 6$

Continuing with the concept of DNL it is reached another of the important issues in the converters, the Missing Codes. As the maximum value of DNL is the deviation of the steps in V_{LSB} units, when this value is greater than or equal to 1 (in absolute value) it normally means that there is at least a Missing Code, provided that the slope of the better approximation is similar to 1, otherwise it occurs as in the case of Figure 5.3, where a value of DNL greater than 1 is not due to Missing Codes, but to the fact that the converter is very steep.

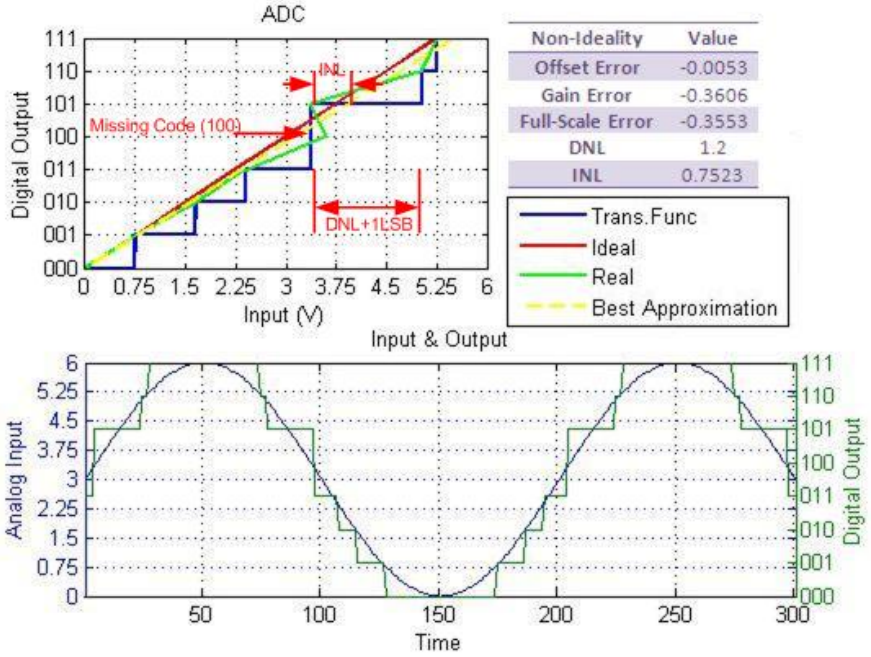


Figure 5.7: *3-Bit Converter with steps* $= [0\ 0.2\ 0.2\ 0.8\ -0.5\ 0.7\ 0\ 0]$, $V_{REF} = 6$

It must also bear in mind that although, it is only performed the representation of the transfer function for the range between 0 and V_{REF}, calculations of non-idealities are done including the values outside this range. For that reason, for DNL and INL, it is returned an array with the values of non-linearity in each interval, so that it can be identified more easily:

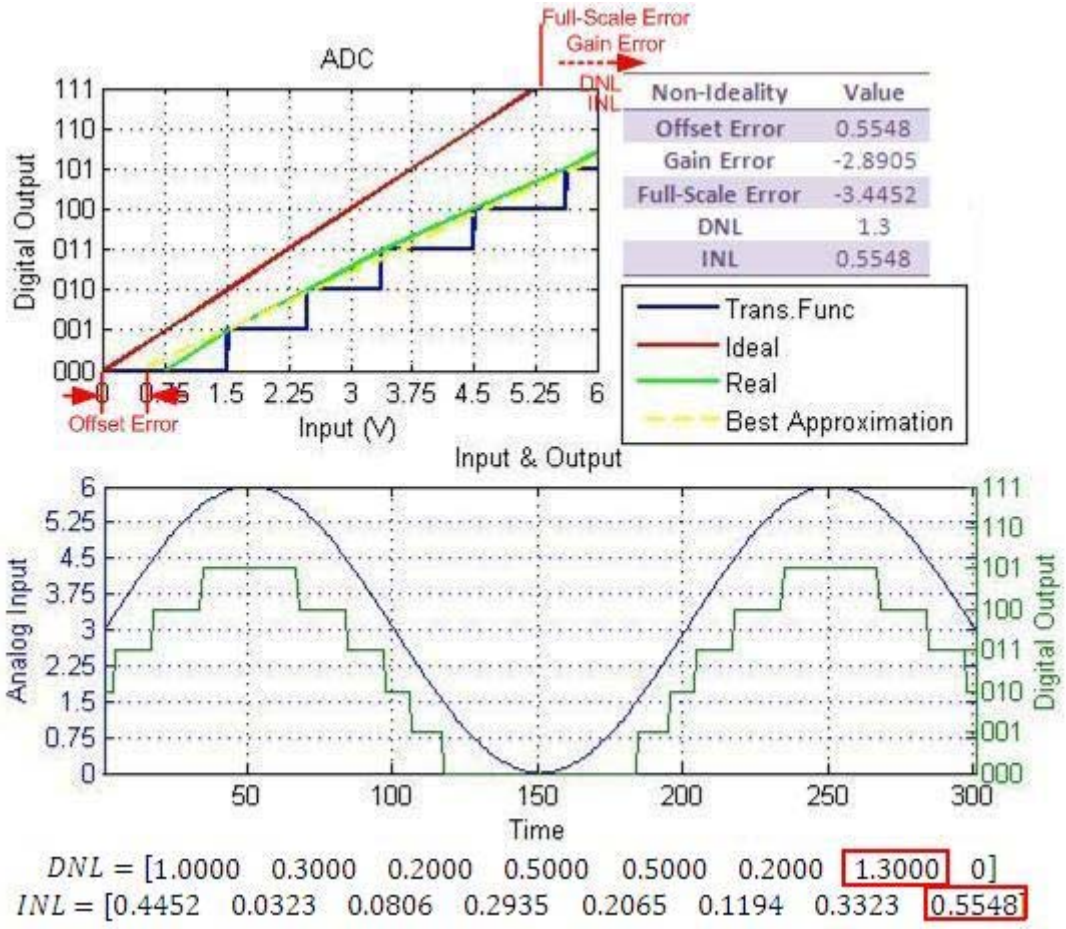


Figure 5.8: *3-Bit Converter with steps* $= [1\ 1.3\ 1.5\ 2\ 2.5\ 2.7\ 4\ 0]$, $V_{REF} = 6$

This is just a sample of all possible combinations that can be obtained varying *steps* and *slopes*, or if instead of that, only one of them is modified, as it is possible to reach the same converters in several different ways. Here there are some other examples:

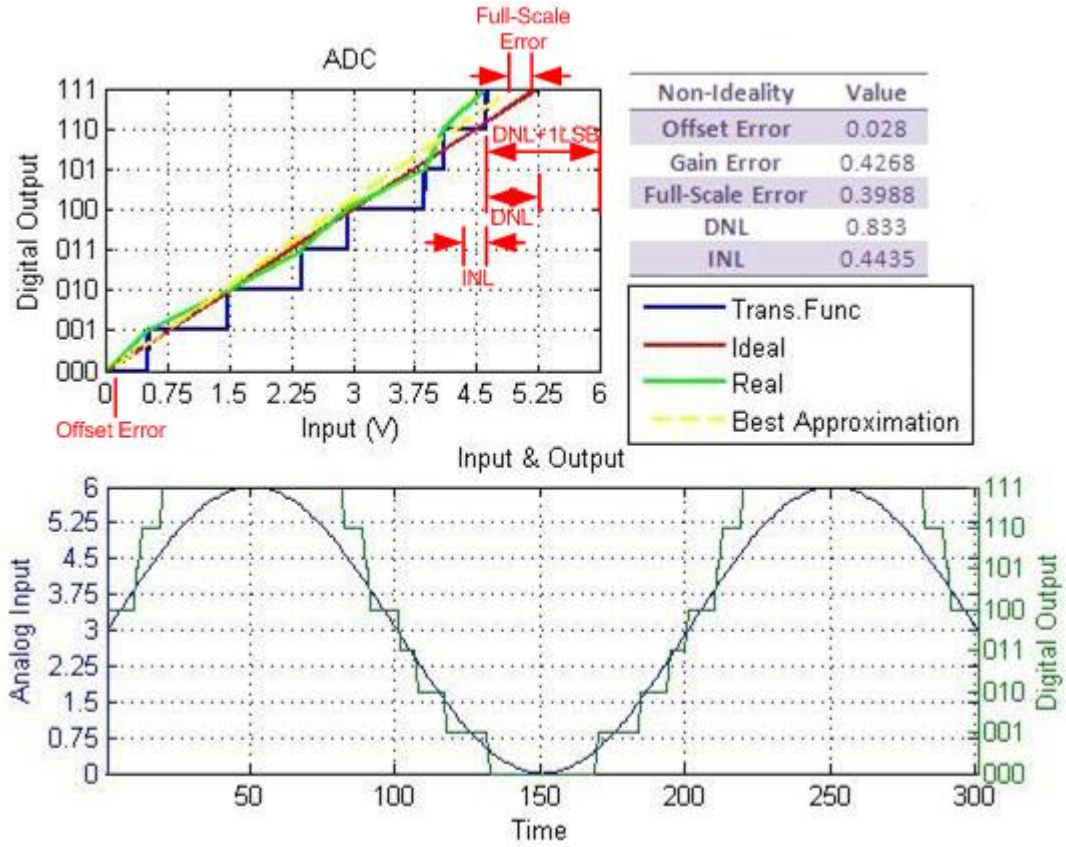


Figure 5.9: *3-Bit Converter with steps* $= [0\ 0.3\ 0.5\ 0\ 0\ -0.2\ 0\ 0]$ *and slopes* $= [1.5\ 1\ 1\ 0.8\ 0.8\ 2\ 2\ 1]$, $V_{REF} = 6$

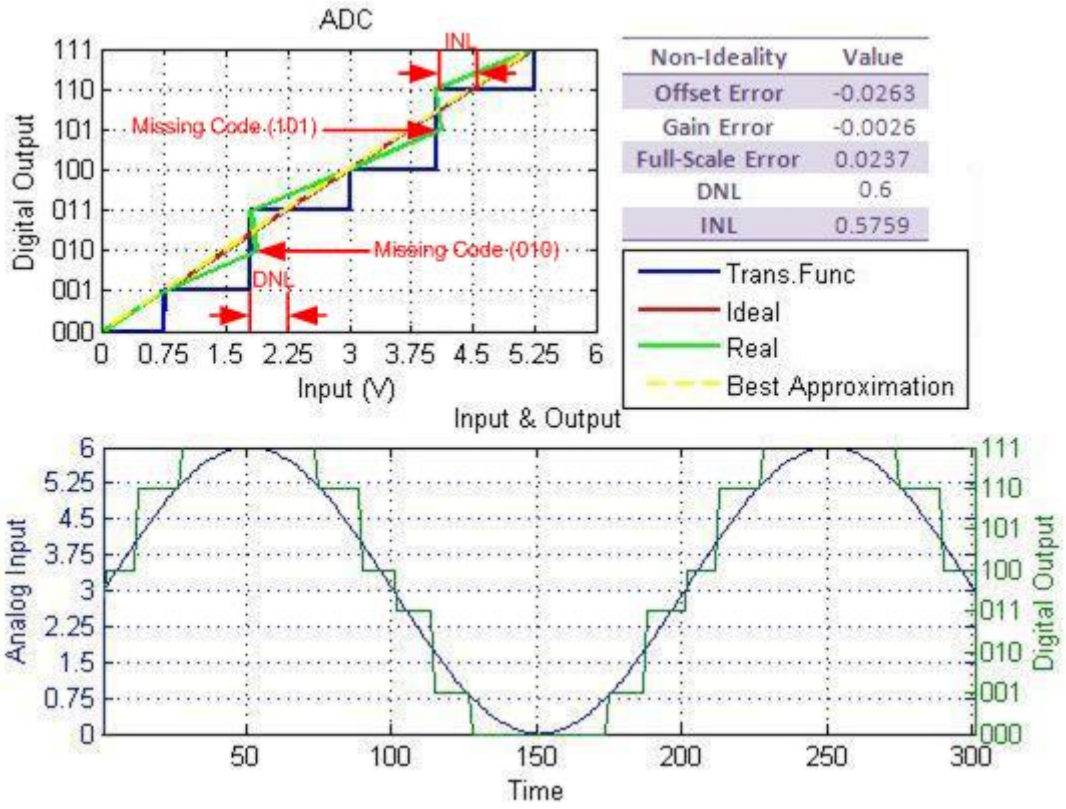


Figure 5.11: *3-Bit Converter with steps* $= [0\ 0.5\ -0.6\ 0\ 0.5\ -0.6\ 0\ 0]$, $V_{REF} = 6$

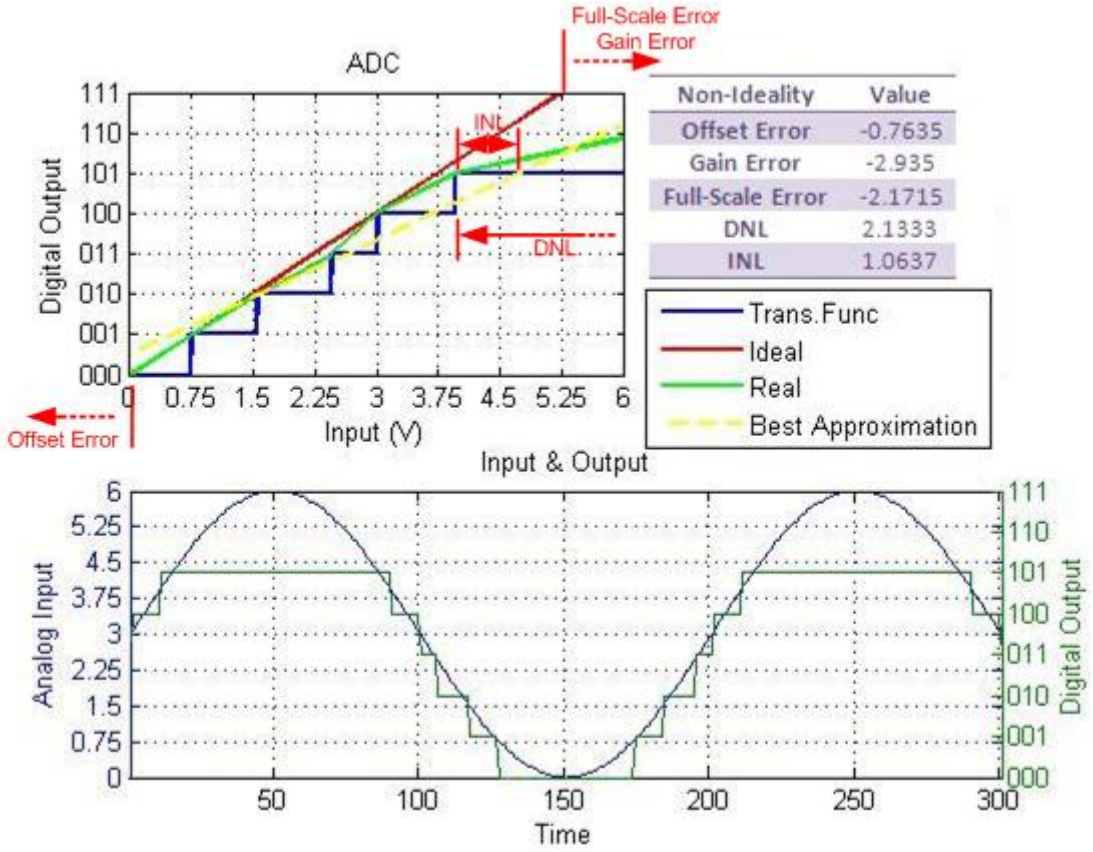


Figure 5.10: *3-Bit Converter with steps* $= [0\ 0.3\ 0.5\ 0\ 0\ -0.2\ 0\ 0]$ *and slopes* $= [1\ 1.3\ 1\ 0.8\ 0.8\ 0.3\ 1\ 1]$, $V_{REF} = 6$

6 GUIDE OF EXECUTIONS

This chapter shows the performed execution to obtain each of the figures of converters that have been shown along the document, it is intended to facilitate tracing and understanding of the performed steps.

Figure 4.2

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,0,1, 'simple');
```

Figure 4.4

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,-0.5,1, 'simple');
```

Figure 4.5

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,[0 -0.7 0 -0.5 0.5 0 0.7 0],1,
'simple');
```

Figure 4.7

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,0,2, 'simple');
```

Figure 4.8

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,0,0.5, 'simple');
```

Figure 4.9

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,0,[1.2 1 1.5 1 1 0.5 1 1] ,
'simple');
```

Figure 4.15

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,8,1,1, 'full');
```

Figure 4.19

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,[0 0.5 0 1 0.2 0 -0.4 0],[1 1.3
1 0.5 1 0.9 1 1],'full');
```

Figure 4.23

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,7,[0 0 0 1 -0.2 0 0 0],1,'full');
```

Figure 5.1

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,0,1,'full');
```

Figure 5.2

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,0.75,1.4,'full');
```

Figure 5.3

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,0,1.4,'full');
```

Figure 5.4

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,-0.5,0.85,'full');
```

Figure 5.5

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[0 0 0.3 0.5 -0.3 0.4 0.3
0],1,'full');
```

Figure 5.6

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[0 0 0 -0.2 -0.5 0 0 0],1,'full');
```

Figure 5.7

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[0 0.2 0.2 0.8 -0.5 0.7 0
0],1,'full');
```

Figure 5.8

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[1 1.3 1.5 2 2.5 2.7 4
0],1,'full');
```

Figure 5.9

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[0 0.3 0.5 0 0 -0.2 0 0],[1.5 1
1 0.8 0.8 2 2 1],'full');
```

Figure 5.10

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[0 0.3 0.5 0 0 -0.2 0 0],[1 1.3
1 0.8 0.8 0.3 1 1],'full');
```

Figure 5.11

```
[output,offset,fserror,gerror,dnl,inl]=ADC(sin(pi*(0:0.01:3))+1,3,6,[0 0.5 -0.6 0 0.5 -0.6 0
0],1,'full');
```

REFERENCES

[1] D. Robertson, “The Past, Present, and Future of Data Converters and Mixed Signal ICs: a “Universal” Model”, VLSI Circuits, 2006. Digest of Technical Papers. 2006 Symposium on, pp. 1-4.

[2] W. Kester, Ed., *Data Conversion Handbook*. Newnes, 2005. [Online]. Available: http://www.analog.com/library/analogdialogue/archives/39-06/data_conversion_handbook.html . [Accessed Mar.9, 2010].

[3] “Analog-to-digital converter”, in *Wikipedia*. [Online]. Available: http://en.wikipedia.org . [Accessed Mar.9, 2010].

[4] “Conversión analogical-digital”, in *Wikipedia*. [Online]. Available: http://es.wikipedia.org . [Accessed Mar.9, 2010].

[5] E. J. Mastascusa, “Analog to Digital Converters (A/Ds)”. [Online]. Available: http://www.facstaff.bucknell.edu/mastascu/eLessonsHtml/Interfaces/ConvAD.html . [Accessed Mar.9, 2010].

[6] N. Gray, “ABCs of ADCs – Analog-to-Digital Converters Basics”. [Online]. Available: http://www.national.com/appinfo/adc/files/ABCs_of_ADCs.pdf . [Accessed Mar.9, 2010].

[7] R. v. d. Passche, *CMOS Integrated Analog-To-Digital and Digital-To-Analog Converters*, 2nd ed. Boston: Kluwer Academic Publishers, 2003.

[8] D. Kollár, “Pulse processing and Analyses”. [Online]. Available: http://www.dnp.fmph.uniba.sk/~kollar/je_w/el3.htm . [Accessed Mar.9, 2010].

[9] E. Sánchez-Sinencio, “Data Converters”. [Online]. Available: http://amesp02.tamu.edu/~sanchez/ADC-Lecture1.PDF . [Accessed Mar.9, 2010].

[10] P. Hoffman and M. Szmulewicz, “Investigación y Análisis,” in *Osciloscopio USB*, M.S. thesis. [Online]. Available: http://pablohoffman.com/cgi-bin/twiki/bin/view/Oscusb/WebHome . [Accessed Mar.9, 2010].

[11] B. A. Olshausen, “Aliasing,” October 2000. [Online]. Available: https://redwood.berkeley.edu/bruno/npb261/aliasing.pdf . [Accessed Mar.9, 2010].

[12] W. Kester, “Which ADC Architecture Is Right for Your Application?”. [Online]. Available: http://www.analog.com/library/analogdialogue/archives/39-06/architecture.pdf . [Accessed Mar.9, 2010].

[13] “*An Outline of the History of the Transistor,” Transistorized!*, 1999. [Online]. Available: http://www.pbs.org/transistor/album1/index.html . [Accessed Mar.9, 2010].

[14] A. N. Saxena, “Monolithic Concept and the Inventions of Integrated Circuits by Kilby and Noyce,” *Nano Science and Technology Institute (NSTI)*, May 2007. [Online]. Available: http://www.nsti.org/Nanotech2007/WCM2007/Saxena.pdf . [Accessed Mar.9, 2010].

[15] H. Veendrick, *Deep-submicron CMOS ICs: from basics to ASICs*, 2nd ed. Reading, 2000. [E-book] Available: Google Books.

[16] C. A. Jansson, “Bubble Handling A/D Converter Calibration,” European Patent 1374408, October 17, 2002.

ANNEXES

ANNEX I (ADC.m)

```
function [out offset FSError GError DNL INL] =
ADC(vin,nbits,vref,steps,slopes,plotting)
% ADC Analog-to-digital converter.
%   Calculate the digital output and non-idealities according to the
analog
%   input and other parameters.
%   INPUTS:
%    - vin = Analog input signal. All values must be non-negative.
%    - nbits = Number of bits for the digital conversion.
%    - vref = Maximum output value (output values are between 0 and
vref).
%    - steps = Relative offset (in Vlsb respect to the ideal case
%      threshold) for each step, can be a single value (all the steps
have
%      the same offset) or a vector with 2^nbits values. Can be both
%      positive or negative.
%    - slopes = Slope for each step of the converter, like 'steps' can
be a
%      single value or a vector, the value must be greater than '0'.
%    - plotting = Kind of plotting. 'simple' for just the transfer
function
%      of the converter, or 'full' for all the data.
%
%   OUPUTS:
%    - output = Digital output signal.
%    - offset = Offset error. Referred to the best linear
approximation of
%      the converter's function.
%    - FSError = Full-scale error. Referred to the best linear
%      approximation of the converter's function.
%    - GError = Gain error. Referred to the best linear approximation
of
%      the converter's function.
%    - DNL = Differential Non-Linearity. Referred to the best linear
%      approximation of the converter's function.
%    - INL = Integral Non-Linearity. Referred to the best linear
%      approximation of the converter's function.
%
%   Example call: ADC(sin(pi*(0:0.01:3))+1,3,8,0.5,[2 1.2 1 1.3 0.7 0
1 1]);
%
%   See also ADCPLOTTING, ADCPLOTTINGSIMPLE.
close all;
% Vlsb value
vlsb = vref/2^nbits;
% All values are in Vlsb
% ADC's ideal transfer function
idealConverter = (1:2^nbits)*vlsb;
% ADC's real transfer function
converter = zeros(2,2^nbits);
% Thresholds of conversion
slope_steps = ((ones(1,2^nbits)./slopes)-
1)*triu(ones(2^nbits,2^nbits));
converter(1,:) = ((1:2^nbits)+slope_steps+steps)*vlsb;
```

```
% Digital output values
converter(2,:) = (1:2^nbits)-1;
% ADC's transfer function's linear interpolation
approxFunction = polyfit([steps(1) converter(1,1:end-
1)/vlsb],converter(2,:),1);
approxConverter = (converter(2,:)-
approxFunction(2))*vlsb/approxFunction(1);
% Output info
offset = approxConverter(1)/vlsb;
FSError = (idealConverter(end-1)-approxConverter(end))/vlsb;
GError = FSError+offset;
aux_inl = converter(1,1)-vlsb/slopes(1);    % INL Adjustment
INL = abs(approxConverter-[aux_inl converter(1,1:end-1)])/vlsb;
DNL = ([converter(1,1:end-1) vref]-[0 converter(1,1:end-1)])/vlsb-1;
DNL(DNL<=-1) = 0;   % DNL Adjustment


% Readjust negative thresholds
R = find(converter(1,:)<=0,1,'last');
converter(1,1:R)=0;
% Auxiliary (2^nbits)x(length(vin)) matrix with the square of each
digital
% value in the columns
aux = vin*0+1;
% Weight up the new matrix with the reference voltage and the number
of
% bits
aux = (converter(1,:)'.^2*aux);
% Weight up the input and compare it with the auxiliary matrix
conversion = converter(1,:)'*(vin*vref/2);
conversion = conversion>aux;
% After fixing the problem of bubbles the output will be the number of
ones
% in each column
out = max(((1:2^nbits)'*ones(1,length(conversion))).*conversion);
% Compensation of the 0s for non-positive thresholds
if R>0
    out(out==0) = R;
end
% Limit maximum output value
out(out>2^nbits-1) = 2^nbits-1;
% Plotting
if strcmp(plotting,'full')

ADCplotting(vin,out,nbits,vref,idealConverter,converter,approxConverte
r,vlsb,slopes(1));
elseif strcmp(plotting,'simple')
    ADCplottingSimple(nbits,vref,converter,vlsb);
else disp('Wrong plotting input value');
end
```

ANNEX II (ADCplotting.m)

```
function
ADCplotting(vin,out,nbits,vref,idealConverter,converter,approximation,
vlsb,slope)
% ADCPLOTTING ADC plot.
%   Plot the transfer function, input and output of the ADC.
%   INPUTS:
%    - vin = Analog input signal. All values must be non-negative.
%    - out = Digital output signal.
%    - nbits = Number of bits for the digital conversion.
%    - vref = Maximum output value (output values are between 0 and
vref).
%    - idealConverter = Threshold values of the ideal converter.
%    - converter = Threshold values of the real converter.
%    - approximation = Threshold values of the best linear
approximation
%      converter.
%    - vlsb = Vlsb value.
%    - slope = Slope value
%
%   See also ADC, ADCPLOTTINGSIMPLE.

% Number of points in the plot
voltPoints = 2^14;
converterRep = zeros(1,voltPoints);
% Axis labels
inScale = 0:vlsb:vref;
outScaleString = dec2bin((0:(2^nbits)-1),nbits);
% Representation of the ADC transfer function
scaling = voltPoints/vref;
scaledConverter = converter(1,:)*scaling;
scaledConverter(scaledConverter<0) = 0;
for i = 2:2^nbits+1
    converterRep(round(scaledConverter(i-1))+1:end) = i-1;
end
converterRep(converterRep>2^nbits-1) = 2^nbits-1;
% Used in plotting
idealTransFunc = zeros(1,2^nbits);
idealTransFunc(2:end) = idealConverter(1,1:end-1)*scaling;
transFunc = [scaledConverter(1,1)-((1/slope)*vlsb*scaling)
scaledConverter(1,1:end-1)];
% Plotting transfer function
subplot(2,1,1), plot(converterRep,'LineWidth',2);
title('ADC');
xlabel('Input (V)');
ylabel('Digital Output');
grid on;
set(gca,'XLim',[0 voltPoints]);
set(gca,'XTick',(0:2^nbits)*vlsb*scaling);
set(gca,'XTickLabel',inScale);
set(gca,'YLim',[0 2^nbits-1]);
set(gca,'YTick',0:2^nbits);
set(gca,'YTickLabel',outScaleString);
hold on;
plot(idealTransFunc,converter(2,:),'r','LineWidth',2);
plot(transFunc,converter(2,:),'g','LineWidth',2);
plot(approximation*scaling,converter(2,:),'y--','LineWidth',2);
```

```
legend('Trans.Func','Ideal','Real','Best
Approximation','Location','SouthEastOutside');
% Plotting signals
x = 1:length(vin);
subplot(2,1,2), AX = plotyy(x,vin*vref/2,x,out);
title('Input & Output');
xlabel('Time');
ylabel(AX(1),'Analog Input');
ylabel(AX(2),'Digital Output');
set(AX,'XLim',[1 length(vin)]);
set(AX(1),'YLim',[0 vref]);
set(AX(1),'YTick',inScale);
set(AX(1),'YTickLabel',inScale);
set(AX(2),'YLim',[0 2^nbits-1]);
set(AX(2),'YTick',0:2^nbits);
set(AX(2),'YTickLabel',outScaleString);
set(AX,'XGrid','on');
set(AX(1),'YGrid','on');
```

ANNEX III (ADCplottingSimple.m)

```
function ADCplottingSimple(nbits,vref,converter,vlsb)
% ADCPLOTTINGSIMPLE ADC simple plot.
%   Plot the transfer function of the ADC.
%   INPUTS:
%    - nbits = Number of bits for the digital conversion.
%    - vref = Maximum output value (output values are between 0 and
vref).
%    - converter = Threshold values of the real converter.
%    - vlsb = Vlsb value.
%
%   See also ADC, ADCPLOTTING.

% Number of points in the plot
voltPoints = 2^14;
converterRep = zeros(1,voltPoints);
% Axis labels
inScale = 0:vlsb:vref;
outScaleString = dec2bin((0:(2^nbits)-1),nbits);
% Representation of the ADC transfer function
scaling = voltPoints/vref;
scaledConverter = converter(1,:)*scaling;
scaledConverter(scaledConverter<0) = 0;
for i = 2:2^nbits+1
    converterRep(round(scaledConverter(i-1))+1:end) = i-1;
end
converterRep(converterRep>2^nbits-1) = 2^nbits-1;
% Plotting transfer function
plot(converterRep,'r','LineWidth',3);
title('ADC');
xlabel('Input (V)');
ylabel('Digital Output');
grid on;
set(gca,'XLim',[0 voltPoints]);
set(gca,'XTick',(0:2^nbits)*vlsb*scaling);
set(gca,'XTickLabel',inScale);
set(gca,'YLim',[0 2^nbits-1]);
set(gca,'YTick',0:2^nbits);
set(gca,'YTickLabel',outScaleString);
```

ANNEX IV (bubbles_finder.m)

```
function positions = bubbles_finder(in)
% BUBBLES_FINDER
%   Returns the position of the bubbles into a matrix. Remember that
in a
%   (N x M) matrix, position 1 corresponds to the value (1,1),
position N
%   corresponds to (N,1), position 2N to (N,2),...
%   INPUT:
%    - in = Matrix with bubbles.
%
%   OUPUT:
%    - positions = Vector with the positions where there are bubbles.
nbits = log2(length(in(:,1)));
aux = ((2.^(0:2^nbits-1))'*(ones(1,length(in)))).*in;
summation = sum(aux);
max_possible_value = 2*max(aux)-1;
max_possible_value(max_possible_value<0) = 0; % For columns with just
0s
bubbles = max_possible_value-summation;
bubb_value = dec2bin(bubbles);
tamano = size(bubb_value);
bubb_positions = reshape(str2num(bubb_value(:)),tamano);
% Row and column of each bubble.
[column,row] = find(bubb_positions(:,tamano(2):-1:1));
positions = length(in(:,1))*(column-1)+row;
```

ANNEX V (alternative version of ADCplotting.m)

```
function
ADCplotting(vin,out,nbits,vref,idealConverter,converter,approximation,
vlsb,offset,slope)
% ADCPLOTTING ADC plot.
%   Plot the transfer function, input and output of the ADC.
%   INPUTS:
%    - vin = Analog input signal. All values must be non-negative.
%    - out = Digital output signal.
%    - nbits = Number of bits for the digital conversion.
%    - vref = Maximum output value (output values are between 0 and
vref).
%    - idealConverter = Threshold values of the ideal converter.
%    - converter = Threshold values of the real converter.
%    - approximation = Threshold values of the best linear
approximation
%      converter.
%    - vlsb = Vlsb value.
%    - offset = Offset value.
%    - slope = Slope value
%
%   See also ADC.
voltPoints = 1000;
% Axis labels
inScale = 0:vlsb:vref;
outScaleString = dec2bin((0:(2^nbits)-1),nbits);
% Representation of the ADC transfer function
idealTransFunc = zeros(1,2^nbits);

vrefAux = vref;
if vref<1
    converter(1,:) = converter(1,:)/vref;
    idealConverter = idealConverter/vref;
    approximation = approximation/vref;
    vlsb = vlsb/vref;
    vrefAux = 1;
end
converterRep = zeros(1,vrefAux*voltPoints);
% Representation of the transfer function
for i = 1:2^nbits
    if converter(1,i) > 0
        if i==1
            if offset>=0

converterRep(round((offset*vlsb*voltPoints:converter(1,i)*voltPoints)+
1)) = converter(2,i);
            else
                converterRep(round((0:converter(1,i)*voltPoints)+1)) =
converter(2,i);
            end
        elseif i == 2^nbits
            [maxV,maxP] = max(converter(1,:));
            if maxP == length(converter)
                converterRep(round((converter(1,i-
1)*voltPoints:vrefAux*voltPoints)+1)) = converter(2,i);
            else
```

```
converterRep(round((converter(1,i)*voltPoints:vrefAux*voltPoints)+1))
= converter(2,i);
            end
        else
            converterRep(round((converter(1,i-
1)*voltPoints:converter(1,i)*voltPoints)+1)) = converter(2,i);
        end
    end
end
% Used in plotting
idealTransFunc(2:end) = idealConverter(1,1:end-1)*voltPoints;
transFunc = [converter(1,1)-((1/slope)*vlsb) converter(1,1:end-
1)]*voltPoints; %tocado para ajustar el primer valor
% Plotting transfer function
subplot(2,1,1), plot(converterRep,'LineWidth',2);
title('ADC');
xlabel('Input (V)');
ylabel('Digital Output');
grid on;
set(gca,'XLim',[0 idealTransFunc(end)+(vlsb*voltPoints)]);
set(gca,'XTick',(0:2^nbits)*voltPoints*vlsb);
set(gca,'XTickLabel',inScale);
set(gca,'YLim',[0 2^nbits-1]);
set(gca,'YTick',0:2^nbits);
set(gca,'YTickLabel',outScaleString);
hold on;
plot(idealTransFunc,converter(2,:),'r','LineWidth',2);
plot(transFunc,converter(2,:),'g','LineWidth',2);
plot(approximation*voltPoints,converter(2,:),'y--','LineWidth',2);
% Plotting signals
x = 1:length(vin);
subplot(2,1,2), AX = plotyy(x,vin*vref/2,x,out);
title('Input & Output');
xlabel('Time');
ylabel(AX(1),'Analog Input');
ylabel(AX(2),'Digital Output');
set(AX,'XLim',[1 length(vin)]);
set(AX(1),'YLim',[0 vref]);
set(AX(1),'YTick',inScale);
set(AX(1),'YTickLabel',inScale);
set(AX(2),'YLim',[0 2^nbits-1]);
set(AX(2),'YTick',0:2^nbits);
set(AX(2),'YTickLabel',outScaleString);
set(AX,'XGrid','on');
set(AX(1),'YGrid','on');
```